NASA Contractor Report 4348

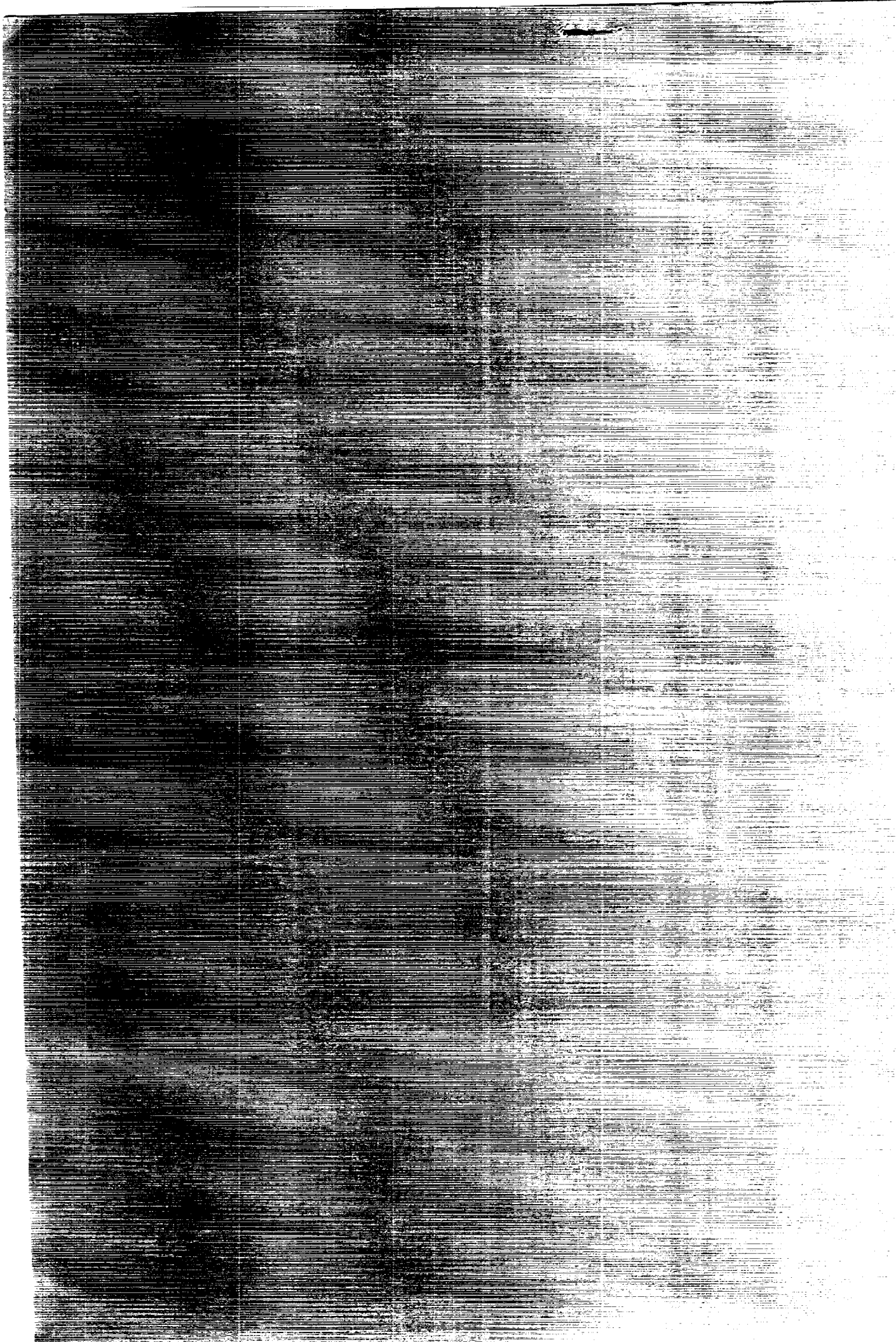
TranAir: A Full-Potential, Solution-Adaptive, Rectangular Grid Code for Predicting Subsonic, Transonic, and Supersonic Flows About Arbitrary Configurations

Theory Document

F. T. Johnson, S. S. Samant, M. B. Bieterman,
R. G. Melvin, D. P. Young, J. E. Bussoletti,
and C. L. Hilmes

CONTRACT NAS2-12513
DECEMBER 1992

NASA



NASA Contractor Report 4348

TranAir: A Full-Potential, Solution-Adaptive, Rectangular Grid Code for Predicting Subsonic, Transonic, and Supersonic Flows About Arbitrary Configurations

Theory Document

F. T. Johnson, S. S. Samant, M. B. Bieterman,
R. G. Melvin, D. P. Young, J. E. Bussoletti,
and C. L. Hilmes
Boeing Military Airplane Company
Seattle, Washington

Prepared for
Ames Research Center
under Contract NAS2-12513

National Aeronautics and
Space Administration

Office of Management

Scientific and Technical
Information Program

1992

Contents

List of Figures

List of Tables

NOMENCLATURE

$C_{i,j}$	=	upwinding operator, see Eqn. (2.43)
DFT	=	discrete Fourier transform
$E(I,J,K)$	=	line moment integral
FFT	=	fast Fourier transform
$\mathcal{F}$	=	nonlinear partial differential operator, see Eqn. (2.1)
$F(I,J,K)$	=	surface moment integral
$F(x)$	=	discrete function representing the boundary value problem
g_1	=	Neumann boundary condition parameter, see Equation (2.4)
g_3	=	Dirichlet boundary condition parameter, see Eqn (2.5)
$\mathcal{G}$	=	continuous Green's function
G	=	discrete Green's function
H	=	total enthalpy
$H(I,J,K)$	=	volume moment integral
I	=	identity matrix
J	=	Bateman functional, see Eqn. (2.9)
L	=	discrete operator for the given boundary value problem
L_0,$\tilde{L}$	=	linearized version of L
M	=	Mach number
N	=	sparse solver preconditioner, or number of grid points
$\hat{n}$	=	unit normal
$\mathcal{O}$	=	symbol designating "order" of magnitude
p	=	static pressure,
q	=	magnitude of the velocity
Q	=	source strength unknowns
r	=	radial distance from the origin
r_p	=	ratio of local total pressure to that at ∞
r_T	=	ratio of local total temperature to that at ∞
R	=	residual, or computational domain (box), or distance away from the boundary
$S_i(\vec{v})$	=	blending function used in upwinding density
$\mathcal{T}$	=	continuous far field (Prandtl-Glauert) operator
T	=	discrete far field operator
$\vec{V}$	=	velocity

$\vec{W}$	=	mass flux vector $\equiv \rho\vec{V}$
α	=	angle of attack, or average operator across a boundary
χ	=	GMRES unknowns
δ	=	discrete delta function, or variation
Δ	=	difference (or jump) across a boundary, or undivided difference
$\vec{\nabla}$	=	gradient, or divergence operator
ϵ	=	small change
γ	=	adiabatic exponent
μ	=	$\Delta\phi$ across a wake, see Eqn. (2.7), also switching function, see Eqn. (2.40)
ν	=	cutoff operator, or edge normal vector
Φ	=	total potential
ϕ	=	perturbation potential
ψ	=	boundary potential
ρ	=	density, see Eqn. (2.2)
Σ	=	boundary or discontinuity surface or summation sign
Ω	=	domain of integration
$\int_{\mathcal{P}}$	=	integral over volume or surface region $\mathcal{P}$
∂R	=	boundary of the domain R

Subscripts

$-$	=	onesided derivative
c	=	cut off value
ps	=	pseudo unknown
x	=	derivative in the coordinate direction x
y	=	derivative in the coordinate direction y
z	=	derivative in the coordinate direction z
∞	=	at infinite distance from the configuration

SUMMARY

A new computer program, called TRANAIR, for analyzing complex configurations in transonic flow (with subsonic or supersonic freestream) has been developed. This program provides accurate and efficient simulations of nonlinear aerodynamic flows about aritrary geometries with the ease and flexibility of a typical panel method program.

The numerical method implemented in TRANAIR is described in this report. The method solves the full potential equation subject to a set of general boundary conditions and can handle regions with differing total pressure and temperature. The boundary value problem is discretized using the finite element method on a locally refined rectangular grid. The grid is automatically constructed by the code and is superimposed on the boundary described by networks of panels; thus no surface fitted grid generation is required. The nonlinear discrete system arising from the finite element method is solved using a preconditioned Krylov subspace method embedded in an inexact Newton method. The solution is obtained on a sequence of successively refined grids which are either constructed adaptively based on estimated solution errors or are predetermined based on user inputs. Many results obtained by using TRANAIR to analyze aerodynamic configurations are presented.

Chapter 1

INTRODUCTION

1.1 MOTIVATION

The role of computational modeling in engineering design has been well recognized for many years. Engineering problems are routinely solved through numerical simulations. In the aerospace industry, for example, aerodynamic flow about aircraft is often simulated using computational tools. Computational Fluid Dynamics (CFD) is rapidly becoming an equal partner with the wind tunnel and flight testing in the design of aerodynamic shapes [1, 2, 3].

Many engineering designs are geometrically complex. In aerodynamics, problems such as the analyses of close-coupled nacelles and high lift systems on a typical transport aircraft configuration (see Figure 1.1) can involve extremely complicated geometries and highly nonlinear flows containing shock waves and convected wakes. The geometry and flow become even more complex for a fighter type aircraft (see Figure 1.2). There is a lack of tools that can routinely handle such complex geometries and treat the appropriate physical phenomena.

Such tools should be reliable, accurate, flexible and efficient. Among the currently available computational tools, panel methods [4]-[14] have long been able to handle complex configurations and boundary conditions in a reliable manner, but they are limited to linear flow models. Aerodynamicists who use panel methods take for granted the ability to add, move, or delete components at will, readily select and change boundary condition types, and obtain accurate solutions at reasonable cost. Multiply connected regions (flap gaps, nacelle interiors), varying length scales, and flow features such as oblique shocks (in supersonic freestream flow), present few problems to a good panel method. Consequently, there are many instances where designers have compromised the physics of the problem and used panel methods in order to try to understand the aerodynamic effects of complex geometry [15],[16]. However, there are other instances involving transonic flows with normal shocks where such compromises are not possible.

The state of the art in calculating transonic flows has progressed significantly since the initial breakthrough by Murman and Cole [17]. Much success has been achieved in solving various forms of the potential equation, the Euler equations, and even the Navier-Stokes equations for special configurations [18]-[35]. Nevertheless,

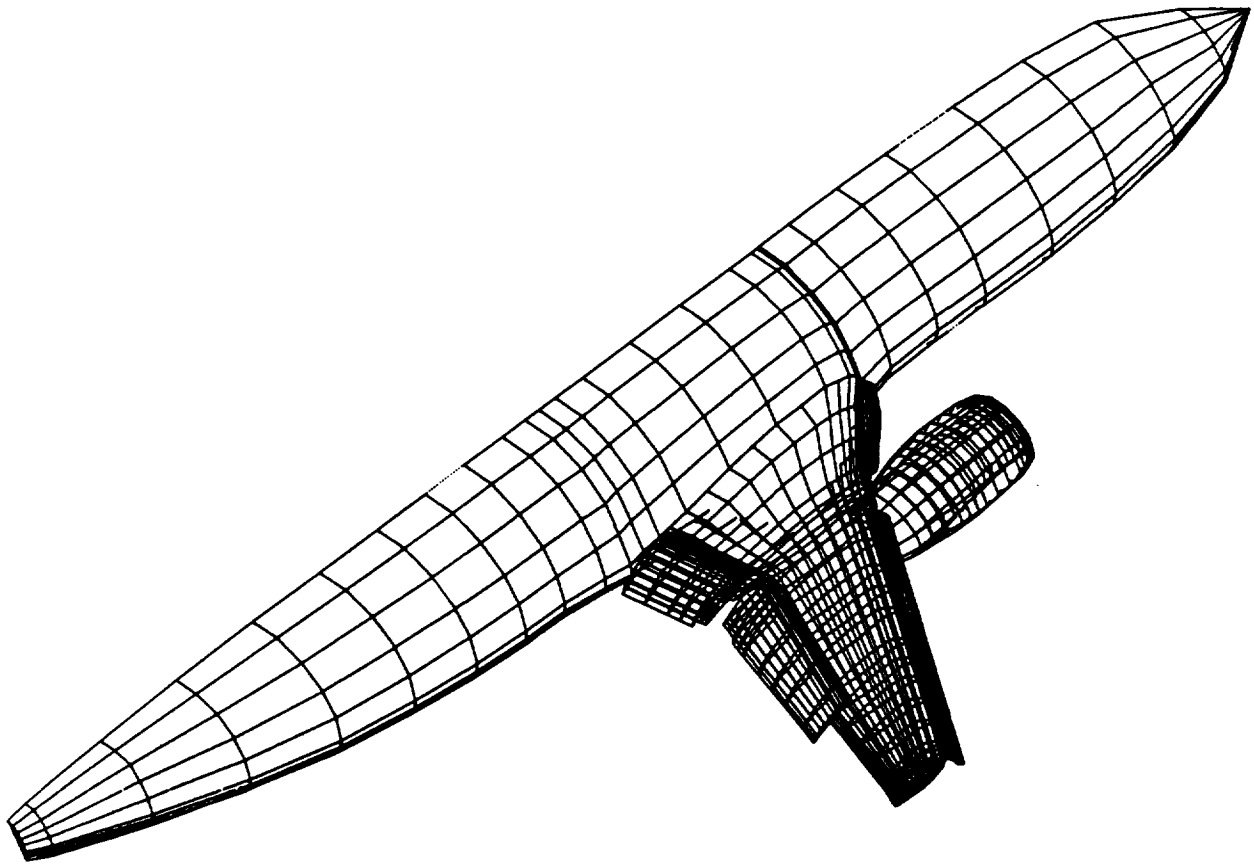


Figure 1.1: Complete Transport Configuration.

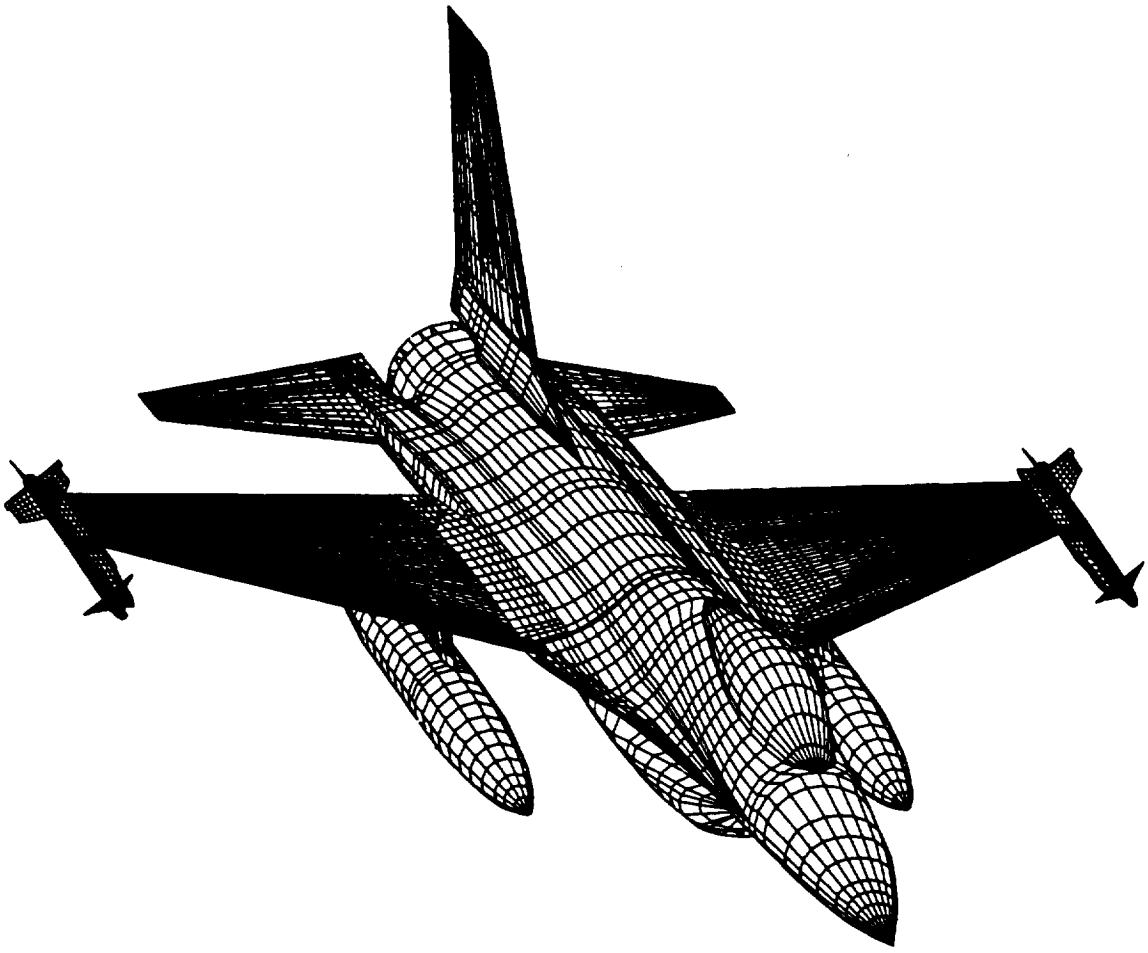


Figure 1.2: Typical Fighter-Type Configuration with Store.

routine analysis of complex configurations using more realistic physics has remained a somewhat distant goal. There are several reasons for this.

First and foremost, most current methods use surface fitted grids. Generation of such grids for complex, multiply connected domains is an extremely difficult task. Although significant progress has been made in grid generation techniques over the last five years, timely treatment of configurations similar to those shown in Figures 1.1 and 1.2 still remain beyond the capabilities of these methods.

Second, current transonic algorithms place severe demands on grids. Few algorithms can adequately handle the anomalies which would result from the application of present grid generation techniques to complex configurations, e.g., non-analytic grids, collapsed edges, fictitious corners, oblique and/or high aspect ratio cells, etc.

Third, many transonic methods are limited to normal flow boundary conditions. Panel methods routinely allow design boundary conditions, porous wall boundary conditions, surface jump conditions, etc. These types of boundary conditions are difficult to implement in the field grid methods.

Fourth, computer run costs can be very high. In the course of airframe design, engineers make hundreds of runs varying angle of attack, Mach number, flap settings, inlet mass ratios, nacelle placement, etc. Thus it is imperative that individual runs be fairly economical. This is clearly a difficult goal to achieve for large configurations, especially when the grid must be fine enough for the reliable prediction of drag increments. For many methods use of additional grid points causes substantial degradation in convergence with corresponding increase in cost.

In this report an approach developed to overcome these problems is described. This approach has been implemented in a computer program called TRANAIR[36]-[51]. It has all the modeling generality and flexibility offered by the PAN AIR technology panel code [10] while solving the nonlinear full potential equation.

The most important feature of this approach is the use of rectangular grids combined with an independently-described configuration definition. The grid is superimposed over the configuration surface. The configuration surface is defined in terms of panels. (see Figure 1.3). This makes the program very easy to use since no surface fitted grids are required. Clearly a rectangular grid can always be superimposed on the configuration regardless of surface topology. But, there are several issues associated with rectangular grids that must be addressed.

First, in order to accurately capture small scale effects, even for relatively simple geometries, local grid refinement is essential. (It should be noted that local grid refinement is also necessary for other approaches that use surface fitted grids for complex configurations, since bunching grid lines is only feasible when there are just a few regions requiring dense grids.) The grids used in TRANAIR are therefore locally refined.

Second, rectangular grids cannot take advantage of a directional difference in length scales in a straightforward manner since high aspect ratio cells skew to the coordinate axes cannot be created by varying grid density. However, this leads to only modest increases in the number of grid points for inviscid solutions if the locally refined grid is located judiciously.

Third, combining a general boundary configuration with a rectangular grid pro-

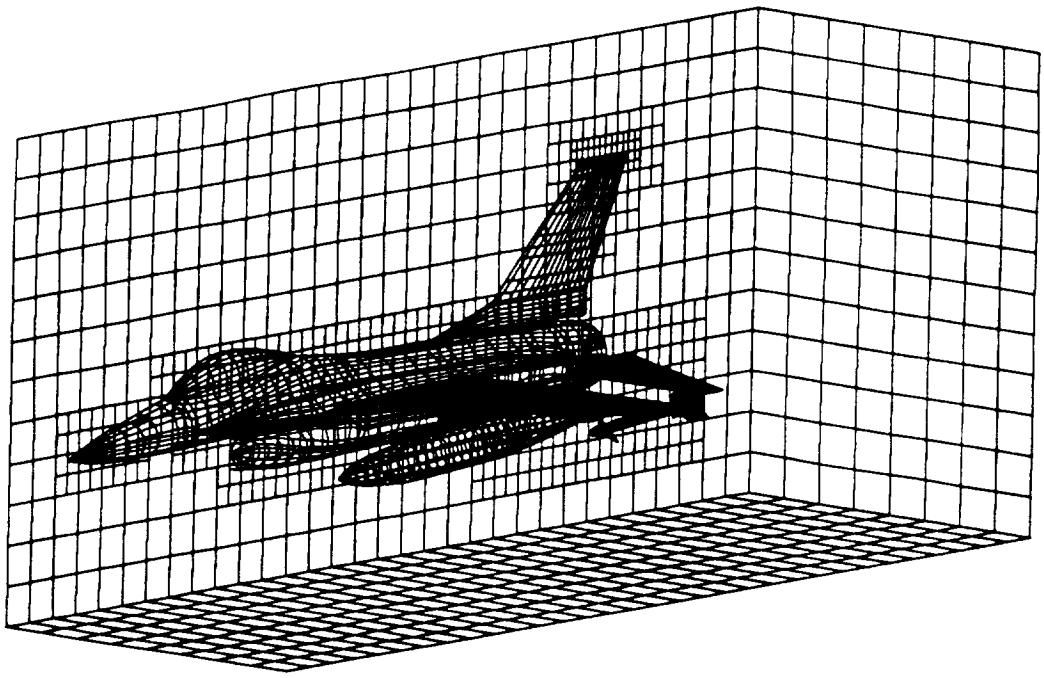


Figure 1.3: TRANAIR Geometry Scheme.

duces irregularly shaped regions near the boundary. The problem of defining accurate discrete equations (operators) at node points adjacent to these regions can be solved by using a finite element method. In TRANAIR the finite element method is implemented using the Bateman variational principle [52]. A generalization of the Bateman principle allows easy implementation of nonstandard boundary conditions. Away from the boundary, application of the finite element method is straightforward except for the treatment of infinite domains. The condition at infinity is satisfied using concepts from integral equation methods whereby potential unknowns are transformed to source unknowns. The inverse transformation is easily accomplished through the use of fast Fourier transforms (FFT's) on rectangular grids.

The finite set of equations thus generated is solved iteratively by an inexact Newton method. At each step of the Newton method the Generalized Minimal RESidual algorithm (GMRES) [53],[54] is used as a driver to solve the linearized problem. GMRES is preconditioned by a Poisson solver and a direct sparse solver. Fast and reliable convergence is achieved by combining these preconditioners in a unique manner. The robustness of the method is achieved through the use of a sequence of refined grids which are either constructed adaptively based on estimated solution errors or are predetermined based on user inputs.

1.2 REPORT ORGANIZATION

This is the Theory Document for the TRANAIR program. In this document the theoretical aspects of TRANAIR are described. A fairly comprehensive description of the method is provided in Chapter 2. Results obtained using the TRANAIR code are described in Chapter 3. In Chapter 4 ideas on future directions are discussed. Many topics which require more detail are discussed in the Appendices. The appendices are oct-tree data structures (Appendix A), implementation of the Bateman variational principle (Appendix B), GMRES algorithm (Appendix C), the exterior Poisson solver (Appendix D), and the sparse solver preconditioner (Appendix E).

Information on how to use TRANAIR is provided in the User's Manual [55]. Specifically, the preparation of input required by the program and the scripts (job control cards) required to run the code on the Cray Y-MP at NASA Ames Research Center (UNICOS) and the Cray X-MP at Boeing (COS) are described.

Chapter 2

METHOD

In this chapter a comprehensive description of the numerical method used in TRANAIR is provided. This numerical method combines diverse component algorithms in an effective manner.

There are eight sections in this chapter. In Section 2.1, the boundary value problem to be solved is presented. In Section 2.2, an outline of the method is provided. In Sections 2.3 and 2.4 the discretization and the solution techniques respectively, are described. In Section 2.5 a postprocessing technique to obtain smooth aerodynamic quantities is presented. In Section 2.6, the extension of the method to problems in supersonic free stream flow is described. In Section 2.7, certain programming considerations are discussed.

2.1 PROBLEM DEFINITION

2.1.1 Governing Equation

The *full potential equation* of aerodynamics is

$$\mathcal{F}(\Phi) \equiv \vec{\nabla} \cdot \rho \vec{\nabla} \Phi = 0 \tag{2.1}$$

where Φ is the *total velocity potential* to be determined, and the density is given by

$$\rho = \rho_\infty \left[1 + \frac{\gamma - 1}{2} M_\infty^2 (1 - \frac{q^2}{q_\infty^2}) \right]^{\frac{1}{\gamma - 1}} \tag{2.2}$$

Here, $q = |\vec{\nabla}\Phi|$ is the local speed, $\vec{V}_\infty$ is the uniform onset flow, $q_\infty = |\vec{V}_\infty|$ is the free stream speed, ρ_∞ is the free stream density, M_∞ is the free stream Mach number, and γ is the ratio of specific heats. Equation (2.1) describes the conservation of mass in inviscid irrotational compressible flow.

2.1.2 Boundary Conditions

Boundary conditions on the configuration surface are required to define a well posed problem. The far field condition is

$$\phi = \mathcal{O}(\frac{1}{R}) \tag{2.3}$$

as $x \rightarrow -\infty$, i.e., upstream of the object. The *perturbation potential* is given by $\phi = \Phi - \Phi_\infty$ where $\vec{\nabla}\Phi_\infty = \vec{V}_\infty$.

A wide variety of boundary conditions may be specified on the aircraft configuration. *Normal mass flux* may be specified via

$$\rho\frac{\partial\Phi}{\partial n} = g_1 \tag{2.4}$$

where n represents the direction normal to the surface. On an impermeable surface g_1 vanishes, whereas, on surfaces such the engine inlets nonzero values for g_1 can be specified.

On engine exhaust surfaces, it is possible to impose the Dirichlet condition

$$\Phi = g_3 \tag{2.5}$$

where tangential flow can be prohibited by specifying g_3 to be constant.

Wakes must extend downstream from lifting bodies. These surfaces allow nonzero circulation in potential flow and can be thought of as thin sheets of concentrated vorticity [5]. The boundary conditions on a wake are

$$\hat{n} \cdot \Delta(\rho\vec{\nabla}\Phi) = 0 \tag{2.6}$$

and

$$\Delta p = 0 \tag{2.7}$$

where

$$p = p_\infty \left[1 + \frac{\gamma - 1}{2} M_\infty^2 (1 - \frac{q^2}{q_\infty^2})\right]^{\frac{\gamma}{\gamma-1}} \tag{2.8}$$

$\hat{n}$ is the unit normal vector, and Δ represents the jump across the wake surface. Equation (2.6) is an expression of conservation of mass across the wake. Equation (2.7) is required for conservation of normal momentum. Equation (2.7) is often linearized about the free stream pressure $p = p_\infty$ assuming small perturbation velocity $\vec{\nabla}\phi$. This leads to the Dirichlet condition that $\Delta\Phi$ is constant along the wake in the direction of $\vec{V}_\infty$. The circulation μ at the trailing edge is determined by imposing a Kutta condition there (see Appendix B.3).

2.1.3 Variational Formulation

The full potential equation may also be derived from the Bateman variational principle [52], namely, that the integral of pressure over the flow field

$$J = \int_\Omega p \, d\Omega \tag{2.9}$$

is stationary. This principle can be used to derive finite element formulas for the full potential equation, (see Section 2.3 below). Taking a variation of J in Eqn. (2.9) and using

$$\frac{\partial p}{\partial \vec{V}} = - \vec{W} \equiv -\rho \vec{V} \tag{2.10}$$

it can shown that

$$\delta J = \int_\Omega - \vec{W} \cdot \delta \vec{V} \, d\Omega \tag{2.11}$$

Integrating by parts,

$$\delta J = \int_\Omega \vec{\nabla} \cdot \vec{W} \;\; \delta\Phi \, d\Omega - \int_\Sigma \hat{n} \cdot \vec{W} \;\; \delta\Phi \, d\Sigma \tag{2.12}$$

where Σ is the boundary of the domain or surface of discontinuity with unit normal $\hat{n}$. (The second integral on the right applies to both sides of Σ in the case of a surface of discontinuity.) If J is stationary with respect to arbitrary variations in Φ, the first integral on the right of Eqn. (2.12) yields the mass conservation equation

$$\vec{\nabla} \cdot \vec{W} = 0 \tag{2.13}$$

This equation is identical to Eqn. (2.1). The second integral on the right yields conservation laws for surfaces of discontinuity. For a shock surface across which mass is continuous it follows that

$$\Delta(\hat{n} \cdot \vec{W}) = 0 \tag{2.14}$$

For a slip surface across which Φ may be discontinuous, $\hat{n} \cdot \vec{W}$ must vanish on both sides. The discontinuity in Φ is determined by Eqn. (2.7).
The natural boundary condition for Eqn. (2.9) may be deduced from Eqn. (2.12), i.e.,

$$\hat{n} \cdot \vec{W} = 0 \tag{2.15}$$

A generalization of the Bateman variational principle which incorporates the boundary conditions described above is that the functional

$$\begin{aligned} J &= \int_\Omega p \, dV + \int_{\partial\Omega_1} g_1 \Phi dS \\ &- \int_{\partial\Omega_2} \alpha(\rho \frac{\partial \Phi}{\partial n})(\Delta\Phi - \mu) dS \\ &+ \int_{\partial\Omega_3} \rho \frac{\partial \Phi}{\partial n}(\Phi - g_3) dS \end{aligned} \tag{2.16}$$

is stationary. Here, g_1 is the specified mass flux on $\partial\Omega_1$, $\Delta\Phi$ is the jump in Φ across the wake surface $\partial\Omega_2$, α denotes the average of the upper surface and lower surface values, and g_3 is the specified potential on $\partial\Omega_3$. The unknown μ represents the jump in Φ on $\partial\Omega_2$ and is determined by Eqn. (2.7).

2.1.4 Regions with Differing Total Properties

A minor modification of the above formulation allows the simulation of flows involving regions of differing *total temperature* and *total pressure*. The flow in each separate region is still potential as long as total temperature and pressure are constant in the region, but pressure and density must be redefined in the following way:

$$p = p_\infty r_p \left[1 + \frac{\gamma-1}{2} M_\infty^2 (1 - \frac{q^2}{q_\infty^2 r_T})\right]^{\frac{\gamma}{\gamma-1}} \tag{2.17}$$

$$\rho = \rho_\infty \frac{r_p}{r_T} \left[1 + \frac{\gamma-1}{2} M_\infty^2 (1 - \frac{q^2}{q_\infty^2 r_T})\right]^{\frac{1}{\gamma-1}} \tag{2.18}$$

Here, r_p is the ratio of the total pressure in the region to the free stream total pressure and r_T is the ratio of total temperature in the region to free stream total temperature. The regions are assumed to be separated by fixed wake surfaces on which two jump boundary conditions are applied. The first is the standard static pressure continuity condition Eqn. (2.7). If the total pressure and/or temperature differences across the wake are large, the pressure formula, Eqn. (2.7) cannot be linearized, i.e., μ can not be assumed to be constant in the downstream direction. The second condition is similar to Eqn. (2.6) but requires a modification to make the answer less sensitive to wake position when total pressure and temperature differences are large. Equation (2.6) is replaced by

$$\hat{n} \cdot \Delta \vec{W}^* = 0 \tag{2.19}$$

where

$$\vec{W}^* = \frac{\rho_\infty q_\infty}{\rho_0 q_0} \rho \vec{\nabla} \Phi \tag{2.20}$$

Here, q_0 is the speed which makes $p = p_\infty$ in the given region and ρ_0 is the density at this speed. Equation (2.6) becomes a natural jump boundary condition for $\Phi^* = q_\infty \Phi / q_0$ if the Bateman principle is modified so that

$$J = \int_\Omega p^* dV \tag{2.21}$$

where

$$p^* = p \frac{\rho_\infty q_\infty^2}{\rho_0 q_0^2}. \tag{2.22}$$

Using this feature, exhaust from engines can be modeled as long as the exhaust can be divided into a finite number of regions each of which has a constant entropy. Section 3.2.5 gives an example of this type of modeling.

2.2 OUTLINE OF THE METHOD

An overview of the numerical method in TRANAIR is shown in Figure 2.1.

TRANAIR uses a locally refined rectangular grid. This grid is generated (Section 2.3.3) in a *computational domain* which extends only as far as is required to enclose all configuration surfaces and any nonlinear flow (Section 2.3.2). First a uniform *global grid* is constructed over the rectangular computational region. Grid cells in the global grid are refined by subdividing a given cell into eight similar cells. The decision to refine or not is controlled by two criteria. In the first specified minimum and maximum cell sizes are used. In the second the density of paneling used to define the surface geometry is used. The size of the local grid box is forced to lie within these two limits. Denser grid is automatically generated where the panels are smaller. Both these forms of control can be exercised over a global region or over certain specially prescribed hexahedral regions. Thus, it is possible to specify arbitrary local hierarchical refinement.

The solution process is carried out over a sequence of grids. This sequence of grids is either predetermined by successively derefining the above constructed grid (in the *grid sequencing option*); or is constructed adaptively (in the *solution adaptive grid option*) where the solution is started on the coarsest grid in the above sequence and the subsequent grids are constructed based on estimates of solution error.

The process of obtaining a solution on each grid is essentially the same in either option and involves two steps. The first involves discretization of the continuous problem and the other involves the solution of the discrete equations.

In the discretization step, the field unknowns Φ are defined at the eight corners of each grid cell and the potential in the cell is defined via *trilinear basis functions* (Section 2.3.4). The unknown parameters are supplemented by boundary unknowns ψ which are values of potential extrapolated across a boundary, and wake unknowns μ which are introduced to satisfy the normal momentum jump condition Eqn. (2.7).

The Bateman variational principle is used to accurately discretize the continuous problem (Section 2.3.4). In particular, this discretization is flux conservative. Special discrete operators are generated for the unknowns near the boundary.

The nonlinear discrete equations are solved iteratively using a Newton method (Section 2.4.2). An initial guess required to start the Newton method is obtained by interpolating the solution on a coarse grid to the next finer one; except on the coarsest grid, where the perturbation potential ϕ is taken to be exactly zero.

For complex configurations involving correspondingly complex physical phenomena, it is advantageous to use more than one technique to solve the algebraic equations. Each technique by itself might reduce errors more rapidly in some subsets of physical and frequency space than in others. Hence it is desirable to treat these techniques as "preconditioners" for an overall convergence stabilization and acceleration scheme such as GMRES (Section 2.4.1). The operators from the finite element method are used to compute residuals and the Jacobian matrix which is inverted using a sparse matrix solver and used as one of the preconditioners (see Appendix E). The iteration is stopped after the converged solution is obtained on the final grid in the sequence.

1. Generate finest grid based on geometry and user specified controls
2. Extract a sequence of grids from the finest grid by repeatedly coarsening all parts of the grid by one refinement level
3. For each grid (starting with the coarsest) solve the problem
   - 3.1 Generate Boundary Operators
   - 3.2 Generate Green's function for the current global grid
   - 3.3 Interpolate initial solution for the current grid
   - 3.4 Solve the discrete equations using Newton's method
     - 3.4.1 Generate and decompose the Jacobian if needed
     - 3.4.2 Solve the linearized problem via GMRES
       - Compute residuals
       - Combine sparse solver and Poisson solver preconditioners
     - 3.4.3 Compute nonlinear update
     - 3.4.3 Return to 3.4.1 if Newton's Method did not converge
   - 3.5 If solution adaptive gridding used
     - 3.5.1 Compute local error estimates
     - 3.5.2 Compute new grid
   - 3.6 If not on final grid then return to 3.1
4. Extract aerodynamic output

Figure 2.1: Overview of the Numerical Method in TRANAIR.

At the conclusion of the solution process, the potential at each grid point is available. From the potential, a wide variety of information concerning the flow about the configuration may be obtained. Velocities in the field about the configuration may be computed, from which streamlines, Mach contours, density distributions, pressure coefficients, force and moment coefficients about the configuration, etc., can be computed.

2.3 DISCRETIZATION

In this section, the discretization process is described. First, the representation of the surface boundary is described. Next, the restriction of computations to a finite region is justified. Then, the computational grid and its generation are described. This is followed by a description of the finite element operators. Grid interfaces between different levels of refinement, modifications of the Bateman principle, and artificial dissipation are described at the end.

2.3.1 Boundary Representation

In TRANAIR, the boundary is described independently of the volume discretization. The geometry of the configuration is represented by a set of *networks* each consisting of a rectangular array of corner points which form arbitrarily shaped quadrilaterals called *panels* (see Figures 2.2). The panels serve the purpose of limiting the region of integration for the Bateman variational functional. No fundamental unknowns (such as the doublets or sources in linear panel methods) are associated with the panels except for wakes, where a discrete set of doublet unknowns, μ, are defined at various corner points of the wakes.

2.3.2 Finite Computational Domain

The computations are restricted to a finite subset of the infinite space. The restriction to a *finite computational domain* can be justified in the following way.

Suppose that the partial differential operator $\mathcal{F}$ is equal to a constant coefficient differential operator $\mathcal{T}$ everywhere outside a finite rectangular region. Let $\mathcal{G}$ be a *Green's function* for $\mathcal{T}$ such that $\mathcal{T}(\mathcal{G} * Q) = Q$ for all Q (where Q are called sources) and $\Phi = \mathcal{G} * Q + \Phi_\infty$ satisfies the far field condition. Then the original differential equation $\mathcal{F}\Phi = 0$ is equivalent to

$$Q + (\mathcal{F} - \mathcal{T})(\mathcal{G} * Q + \Phi_\infty) = 0. \tag{2.23}$$

Outside the finite rectangular region, $\mathcal{F} = \mathcal{T}$ so that $Q = 0$. Thus, the unknowns Q are confined to a bounded region.

For full potential flow, the far field operator is the *Prandtl-Glauert operator*

$$\mathcal{T}\Phi = (1 - M_\infty^2)\Phi_{xx} + \Phi_{yy} + \Phi_{zz} \tag{2.24}$$

Equation (2.24) is a linearization of the full potential Eqn. (2.1) about $\vec{V}_\infty$.

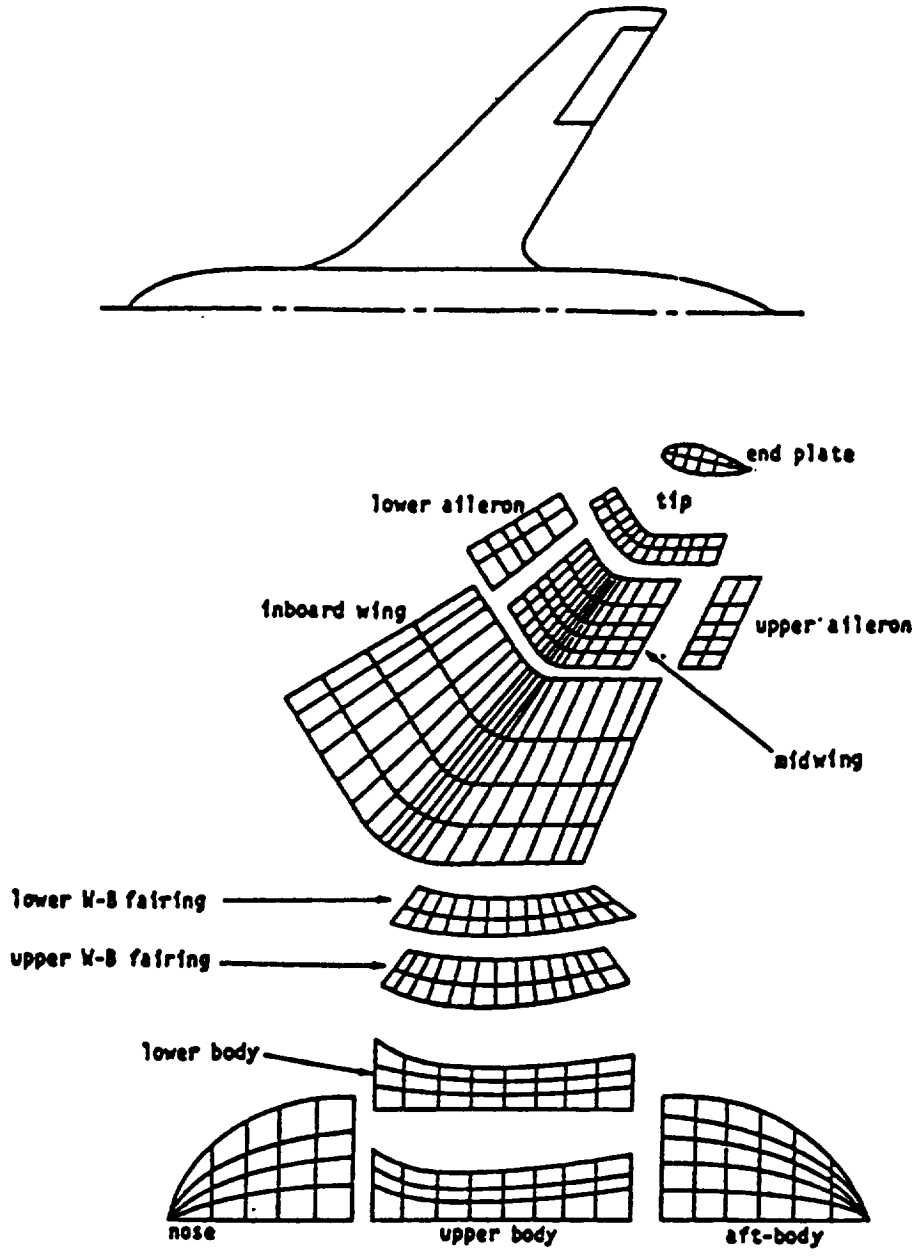


Figure 2.2: Configuration Boundary Description in Terms of Networks of Panels

For most problems Q approaches zero (away from the boundary) much more rapidly than Φ approaches Φ_∞. This enables the termination of the computational domain a short distance away from the boundary or regions of nonlinear flow. Wakes, which extend to infinity, are exceptions which produce sources and sinks that extend to infinity downstream of the configuration. By assuming that such sources and sinks are constant in the downstream direction, their influence can be computed by using a downstream Green's function (see Appendix D).

If the continuous operators are replaced by discrete ones, the same argument holds. The requirement on T, the discrete version of $\mathcal{T}$, is that a discrete Green's function G is available that satisfies an appropriate discrete far field condition and $T(G * Q) = Q$ for all Q. In practice, this means that T is a constant coefficient elliptic operator discretized on a uniform Cartesian grid [36],[39],[89],[90],[94],[96]. Thus, the computational domain need only cover the region where nonlinear flow occurs and where the discrete version L of the operator $\mathcal{F}$ is not approximated well by a discrete far field operator T. The discrete operator T used in the method is the standard seven point finite difference operator.

2.3.3 Computational Grid

The volume grid in the finite computational domain is generated automatically by the code. TRANAIR can operate in one of two modes, either *grid sequencing* (discussed in Section 2.4.3) or *solution adaptive gridding* (described in Section 2.4.4). In either case it is necessary to specify a starting grid. The process for constructing the grid is described below.

First, the finite computational region is chosen to be rectangular and is divided into a coarse uniform rectangular grid, called the *global grid*, which is independent of the boundary surfaces. To facilitate matching between the nonlinear flow inside and the linear flow outside, the global grid includes one plane of unrefined global grid boxes on each face of the computational domain where $L \equiv T$. These boxes remain unrefined and no boundary surface is allowed to cut these boxes (except boxes on the downstream face of the computational domain which can be cut by wakes). This is required because the source Q is assumed to be zero on the boundary points of the global grid.

The global grid is locally refined in a hierarchical manner, i.e., any grid box can be refined into eight geometrically similar boxes of equal volume. This process is repeated to give a grid with any desired local resolution and is controlled by two criteria.

The first criterion for local refinement is based on the length scale of the surface panels used to describe the boundary. Every box element that is sufficiently close to a panel is refined if a weighted length scale (the panel diameter multiplied by a *panel tolerance factor* which is provided as input) associated with the panel is smaller than the length scale associated with that box element. A box is deemed to be in the neighborhood of a panel if a scaled version of it generated by expanding it around its center intersects that panel. The *expansion factor* by which the boxes are scaled can also be specified. This factor is useful if it is desired to extend the effect of the

presence of the boundary on the grid refinement further out into the volume grid. The panel based criterion is effective in providing local refinement near the boundary surface.

The second criterion is the requirement that the *local box size* is restricted to be in a specified range (dx_{min},dx_{max}). Boxes are refined recursively until their size is smaller than dx_{max}. Further refinement depends on the panel based criterion. Refinement based on comparison with dx_{max} is useful in problems where high gradients, such as shock waves, exist in regions away from the boundary surface.

Both criteria for refinement can be invoked over either the entire computation box or over certain *special regions of interest* or *disinterest* where different desired ranges of box sizes and panel tolerance factors are defined. The special regions are hexahedral and provide ample flexibility in generating desired local refinement.

Once all specified refinements are done, the grid is *legalized* so that two boxes abutting on a face or an edge differ by at most one refinement level. This ensures sparse stencils for the finite element operators and simplifies certain data structures [56], but allows sufficiently rapid changes in grid level.

Locally refined box elements formed by the process described above are usually an unstructured collection of box elements. To completely describe these elements an oct-tree data structure is used. The tree is formed as boxes are created through refinement. A box and node numbering system is developed from the tree and adjacency and other information is extracted. (See [41], [57] and Appendix A for more details.)

A grid generated in this manner may be considered to be a prescribed grid. This grid is sequentially derefined to generate a set of coarse to fine grids. If no solution adaptation is used then the iterative solution is obtained on this sequence of grids by starting with the coarsest and moving through the finer grids. If the adaptive method is used then the coarsest grid is used to start the solution and all subsequent grids are determined according to estimated local solution errors. In either case the initial guess for the starting grid is zero and for each subsequent grid is obtained by interpolating the solution from the previous grid.

2.3.4 Finite Element Operators

The discrete operators on every grid in the sequence are constructed using a finite element method. Implementation of the finite element method for rectangular boxes away from the boundary and irregular boxes near the boundary is described below.

Element Trial Functions

Every rectangular element is geometrically identical except for a scale factor. The standard trilinear element trial function, parameterized by eight corner unknowns is used (see Figure 2.3). The trial functions used for elements near the boundary are also represented in a similar manner (see below). In order to generate as compact a stencil as possible (e.g., standard seven point operator for Poisson's equation on a uniform grid) certain lumping terms are added (see Appendix B).

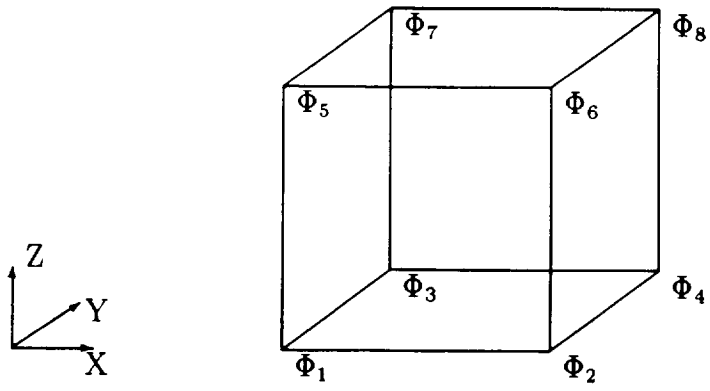


Figure 2.3: Box Finite Element With Eight Corner Unknowns.

Implementation of the Variational Principle

Element stiffness matrices are generated by taking variations of the functional J with respect to each degree of freedom. If only natural boundary conditions are present, (reiterating Eqn. (2.11) and noting that $\vec{W} = \rho\vec{V}$ and $\delta\vec{V} = \nabla\delta\phi$) variation of J are given by

$$\begin{aligned} \delta J &= -\int_\Omega \rho\vec{\nabla}\Phi \cdot \vec{\nabla}\delta\Phi dV \\ &= -\sum_i \int_{\Omega_i} \rho\vec{\nabla}\Phi \cdot \vec{\nabla}\delta\Phi dV \\ &\simeq -\sum_i \rho_i \int_{\Omega_i} \vec{\nabla}\Phi \cdot \vec{\nabla}\delta\Phi dV \end{aligned} \tag{2.25}$$

Here, ρ_i is the value of ρ at the centroid of the elemental region Ω_i. The last step in equation (2.25) is equivalent to replacing ρ by a piecewise constant function. This approximation of the operator coefficients maintains second order accuracy for the potential in the L_2 norm and first order accuracy in the energy norm [58].

Near Field and Far Field Boxes

Near field boxes are boxes not cut by any boundary surface, but where $L \neq T$. Equation (2.25) defines the element stiffness matrices by considering variations of J with respect to each of the eight corner unknowns of the element. Thus, every near field box has the same element stiffness matrix up to a constant factor that depends only on the refinement level of the element and ρ_i. This results in large savings in storage. In addition, ρ is a nonlinear function of the velocity and is evaluated at the centroid of each element during every iteration. Thus, discrete formulas for velocity at the centroids in terms of the unknowns at the corners of the element are needed. Since all near field box elements are similar, only one velocity formula needs to be stored, resulting in additional large savings in storage.

The *far field boxes* lie on faces of the computational domain where $L \equiv T$. These boxes are geometrically identical. Also the density is constant in these boxes since

the linear flow properties are matched. Hence the operators for all such boxes are identical.

Boundary Boxes and D Regions

Boundary boxes are those cut by a boundary surface. A connected subset of a boundary box is referred to as a *D-region*, (see Figure 2.4). Each D-region is bounded by a subset of the boundary surface as well as possibly subsets of the box faces. No D-regions are defined in the interior of the configurations since the flow there is defined to satisfy $\phi = 0$. Hence, a box cut by a boundary surface with flow on one side and no flow (referred to as *stagnation*) on the other would have only one D-region, for example region D_3 in Figure 2.4. It is possible to have more than one D-region in a boundary box. This is the case in Figure 2.4 for D-regions D_1 and D_2 where a wake divides an element.

Since boundary conditions on a surface can induce discontinuities in Φ or $\vec{\nabla}\Phi$, a separate element trial function is needed for each D-region. The element trial function for each such D-region is parameterized by unique unknowns at the corners of the grid box. Corner unknowns on the other side of a boundary surface from their D-region can be viewed as extrapolated values and are denoted by Ψ. In Fig. 2.4 the Ψ_L unknowns correspond to the element trial function in D_2 and the Ψ_U unknowns correspond to the element trial function in D_1. There is a one-to-one correspondence between element trial functions in the box and D-regions.

Stiffness Matrices for D-regions

Each D-region also has a distinct element stiffness matrix that must be stored. The coefficient of the differential operator, ρ, is evaluated at the centroid of the D-region and hence distinct velocity operators are also required for each D-region. However, these boundary elements represent typically only 10 to 20% of the elements needed to give an accurate solution of the boundary value problem.

The element stiffness matrices D-regions are derived from an expanded version of Eqn. (2.25) including appropriate surface integral terms. The domain of integration for the volume integral is the relevant D-region. The domain for the surface integrals is the intersection of the boundary with the boundary of that D-region. Since the integrand is a product of polynomials, volume moments must be computed over the D-region.

Consider the volume moment

$$H(I,J,K) = \int_D x^{I-1} y^{J-1} z^{K-1} dV. \tag{2.26}$$

Since the boundary is parameterized by piecewise flat panels, this moment can be computed exactly via the following procedure. By Gauss' theorem

$$H(I,J,K) = \frac{1}{I+J+K} \int_S x^{I-1} y^{J-1} z^{K-1} (\hat{n} \cdot \vec{R}) dS \tag{2.27}$$

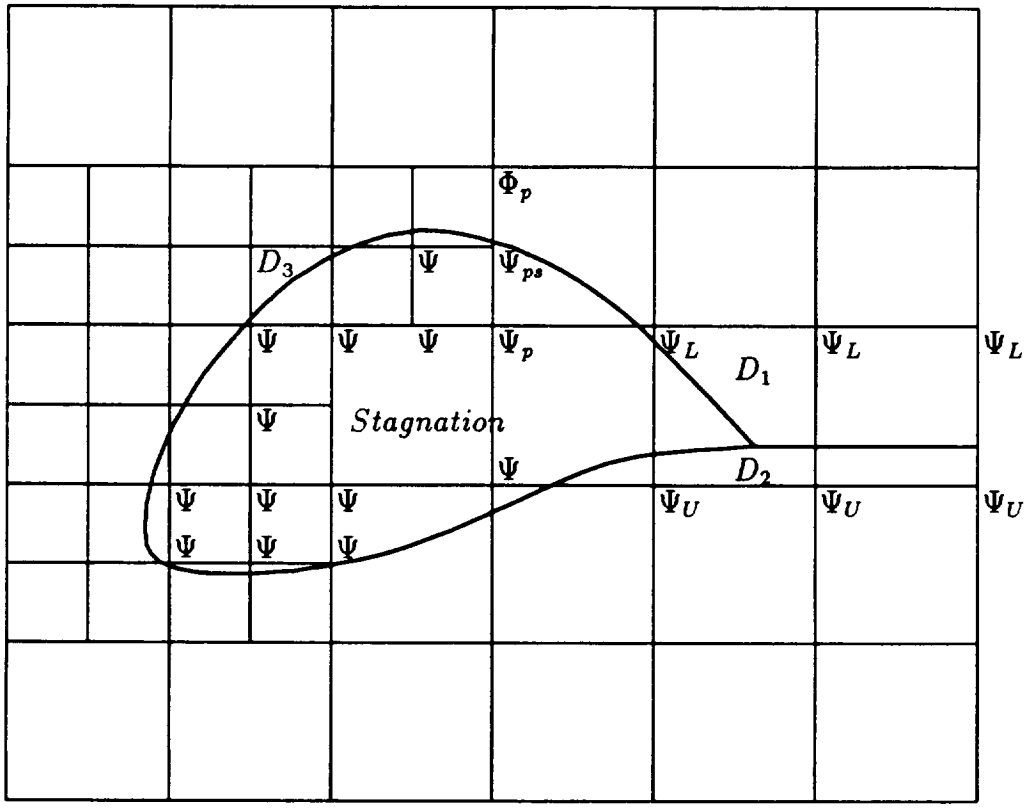


Figure 2.4: Placement of Unknowns. All Grid Points Have Φ Unknowns.

where S is the bounding surface and $\vec{R} = (x, y, z)$. Since S is the union of flat surfaces S_i,

$$H(I,J,K) = \frac{1}{I+J+K} \sum_i F_{S_i} \tag{2.28}$$

where

$$F_{S_i} = n_x F(I+1,J,K) + n_y F(I,J+1,K) + n_z F(I,J,K+1) \tag{2.29}$$

and

$$F(I,J,K) = \int_{S_i} x^{I-1} y^{J-1} z^{K-1} dS \tag{2.30}$$

Each S_i is assumed to be a polygon whose perimeter is the union of straight lines T_{ij}. Using Stokes' theorem, $F(I,J,K)$ for fixed i may be evaluated recursively by the formula

$$F(I,J,K) = \frac{1}{I+J+K-1} \left[(\hat{n} \cdot \vec{R}) F_n + \sum_j E_\nu \right] \tag{2.31}$$

where

$$\begin{aligned} F_n &= n_x(I-1)F(I-1,J,K) + n_y(J-1)F(I,J-1,K) \\ &\quad + n_z(K-1)F(I,J,K-1) \\ E_\nu &= \nu_x E(I+1,J,K) + \nu_y E(I,J+1,K) + \nu_z E(I,J,K+1) \end{aligned}$$

where with $\hat{t}$ denoting the edge tangent vector, $\hat{\nu}$ is the edge normal vector $\hat{t} \otimes \hat{n}$, and $E(I,J,K)$ is defined by

$$E(I,J,K) = \int_{T_{ij}} x^{I-1} y^{J-1} z^{K-1} dl. \tag{2.32}$$

Using simple one dimensional integration formulas $E(I,J,K)$ may be evaluated recursively by the formula

$$E(I,J,K) = \frac{1}{I+J+K-2} \left[E_\zeta + D_t \right] \tag{2.33}$$

where

$$\begin{aligned} E_\zeta &= (I-1)\zeta_x E(I-1,J,K) + (J-1)\zeta_y E(I,J-1,K) \\ &\quad + (K-1)\zeta_z E(I,J,K-1) \\ D_t &= t_x D(I+1,J,K) + t_y D(I,J+1,K) + t_z D(I,J,K+1) \end{aligned}$$

where $\vec{\zeta} = (\hat{t} \otimes \vec{R}) \otimes \hat{t}$ (constant along T_{ij}), and $D(I,J,K) = x^{I-1}y^{J-1}z^{K-1}\,|_1^2$, where 1 and 2 represent the initial and final points of T_{ij}. Thus, the original integrals (2.26) defined over a complicated volume can be systematically reduced to point evaluations at vertices of the bounding surface. The surface moments arising out of the surface integrals in Eqn. (2.17) can be computed starting with Eqn. (2.30). Note that the location of the centroid in each D-region can be computed from the zero and first order moments. Further details on the operator generation including an example of operator calculations are given in Appendix B.

Identifying D-regions

There remains, of course, the problem of identifying D-regions and their bounding surfaces. This is done in three stages.

First, for each panel, a list of grid boxes containing any part of the panel is constructed. Because the grid is rectangular and hierarchical in nature it is relatively easy to isolate the subset of boxes which are located within a neighborhood of a given panel. Moreover, because the boxes are rectangular and the panels are divided into flat triangles it is straightforward to determine if boxes in a neighborhood of a panel in fact contain any part of the given panel. This list is then inverted to find all the panels intersecting a given boundary box.

In the second stage a list of equivalence classes of panel sides for each boundary box is constructed. A panel side is either the upper or lower surface of a panel. An equivalence class consists of all panel sides which are connected to each other through panel edges. A panel side is connected to another panel side if the two panels share a common edge that is partially or wholly contained within the given boundary box and if there is no intervening panel also connected to that edge.

In the third stage separate connected regions of the boundary box are identified. This is done by choosing points on different panel side equivalence classes and then joining them with straight lines. The set of panels cutting these straight lines is examined and the panel side equivalence classes of panels responsible for successive cuts are identified as members of a new equivalence class of panel sides which bound a connected region. Polygonal subsets of a face of the boundary box are included in such an equivalence class whenever a panel side is discovered to intersect the face. This algorithm determines connected regions within a boundary box. However it is also necessary to determine which such regions are connected to regions in adjacent boxes. This is because of the necessity of maintaining continuity of element trial functions across box faces and edges. For this purpose a list of which panels in each boundary box region intersect box faces is stored. These intersections are compared with those in an adjacent box and connections between regions are established. The Ψ parameters at common nodes of connected regions are then identified.

2.3.5 Grid Interfaces

In the finite element method, conservation of mass results if the element trial functions are continuous from box to box. This property can be retained in the presence of

grid refinement by introducing *pseudo-unknowns*. By definition a pseudo-unknown is any unknown located at a node on the boundary of some element but not at a corner of this element. This can occur only at a coarse to fine grid interface. In the two dimensional case, pictured in Figure 2.5, Φ_1 is a pseudo-unknown whose parents are Φ_2 and Φ_3. In the situation pictured in Figure 2.4, Ψ_{ps} is a pseudo-unknown whose parents are Φ_p and Ψ_p.

In order to maintain continuity of the element trial functions across element boundaries, Φ_1 in Figure 2.5 must be the average of its parents, that is

$$\Phi_1 = \frac{1}{2}(\Phi_2 + \Phi_3) \tag{2.34}$$

In three dimensions, pseudo-unknowns can occur at the midpoints of element edges or the centers of element faces. For a pseudo-unknown Φ_1 at the center of an element face with four parents Φ_2,Φ_3,Φ_4, and Φ_5,

$$\Phi_1 = \frac{1}{4}(\Phi_2 + \Phi_3 + \Phi_4 + \Phi_5). \tag{2.35}$$

Thus, pseudo-unknowns are not true degrees of freedom and could be eliminated at the outset from the element stiffness matrices through Equation (2.35). However, this would result in loss of uniformity in these matrices, many special cases, and loss of vectorization. Hence these unknowns are treated as degrees of freedom when the element stiffness matrices are generated. In the process of evaluating the discrete operator L, pseudo-unknowns are first assigned values by averaging their parent unknowns. Residuals of the governing equations are produced at these unknowns but are then distributed to the residuals for their parents. This process of distributing residuals to parents is justified by Eqn. (2.35) and a straightforward application of the chain rule

$$\frac{dJ}{d\Phi_2} = \frac{\partial J}{\partial \Phi_2} + \frac{\partial J}{\partial \Phi_1}\frac{\partial \Phi_1}{\partial \Phi_2} = \frac{\partial J}{\partial \Phi_2} + \frac{1}{4}\frac{\partial J}{\partial \Phi_1}. \tag{2.36}$$

Thus, the component of the residual of the discrete version of Eqn. (2.25) calculated for Φ_1 should be equally distributed to the residuals for the four parent unknowns. This technique has the advantage that every element stiffness matrix produces contributions only to the 8 corner unknowns of its box element, thereby simplifying the generation of the stiffness matrices and enhancing vectorization. Vectorizing over large blocks of similar elements can be done using an outer loop over the eight corner unknowns and an inner loop over the elements in the block.

2.3.6 Modifications to the Bateman Principle

To achieve a stable numerical formulation, the treatment of Dirichlet boundary conditions and wake surfaces must be modified. In addition, the natural Neumann condition must be modified to account for boundary curvature, since the solution is often sensitive to this quantity and the boundary is discretized using flat panels. These modifications are described below.

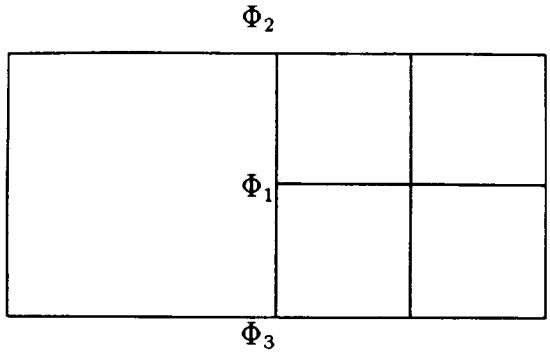


Figure 2.5: Pseudo-Unknown in Two Dimensions.

The introduction of the last surface integral in the variational principle (2.17) enforces a Dirichlet condition on $\partial\Omega_3$. Equation (2.17) can then be used to calculate the element stiffness matrices in a finite element formulation. It turns out that the resultant discrete problem is somewhat unstable. In certain instances one can show that the boundary unknowns Ψ actually satisfy a discrete Helmholtz equation and an oscillatory solution is, in fact, obtained. This phenomenon is probably related to the fact that J is no longer maximized in subsonic flow with Dirichlet data. This suggests a remedy which has been implemented and which has been found to be very reliable numerically. The last integral in Eqn. (2.17) is replaced by

$$\int_{\partial\Omega_3} \left[\rho \frac{\partial \Phi}{\partial n}(\Phi - g_3) - \frac{\rho}{2\Delta l}(\Phi - g_3)^2 \right] dS \tag{2.37}$$

where Δl is the minimum diameter of the box containing the trial function. A similar term may be added to the second integral.

All surfaces are represented by flat panels. Resultant discontinuities in slope from panel to panel will be reflected in the solution as the grid is refined. In most cases, this effect is spurious, since the surface slope discontinuities are artifacts of the panel description of the surface. To eliminate this problem, a curved surface is simulated by adding to the variation of Eqn. (2.17) a surface integral

$$\partial J = \partial J + \int_{\partial\Omega_1} \rho \vec{\nabla}\Phi \cdot (\hat{n} - \hat{n}^*)\partial\Phi dS \tag{2.38}$$

where $\hat{n}^*$ is a polynomial interpolation of $\hat{n}$ and $\partial\Phi$ denotes the variation of Φ. The endpoints for the polynomial interpolation of the normal are user controlled to allow discontinuities in slope where they are actually present.

2.3.7 Dissipation

First order upwinding of the density is used to produce the artificial viscosity required when supersonic flow is present [22, 25]. Such upwinding is given by replacing ρ in

the full potential equation with

$$\tilde{\rho} = \rho - \mu \hat{V} \cdot \vec{\Delta}_{-} \rho \tag{2.39}$$

where $\hat{V}$ is the normalized local velocity and $\vec{\Delta}_{-}\rho$ is an upwind undivided difference. In Eqn. (2.39) μ is the switching function given by

$$\mu = max(1 - M_c^2/M^2, 0) \tag{2.40}$$

where M is the local Mach number and M_c is the cut-off Mach number assigned the value $M_c^2 = 0.95$ chosen to introduce dissipation just below Mach 1.0.

An alternative to upwinding the density is to use flux biasing i.e. upwind the flux ρq where $q = \| \vec{V} \|_2$. Flux biasing may be expressed in a form similar to Eqn.(2.39) by writing

$$\tilde{\rho} = \frac{1}{q}\left((\rho q) - \hat{V} \cdot \vec{\Delta}_{-} (\overline{\rho q})\right) \tag{2.41}$$

where

$$\overline{\rho q} = \begin{cases} \rho q & \text{for } M > 1 \\ \rho^* q^* & \text{for } M \leq 1 \end{cases} \tag{2.42}$$

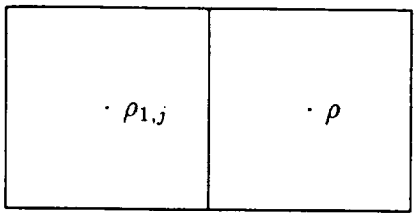
Here, $\rho^* q^*$is the value of ρq at sonic flow conditions.

For either form of dissipation, the upwinding is done across the face of a box with a precomputed stencil. A density or flux is chosen for each of the six faces of an element when the operators are generated. For a uniform grid with no boundaries, each box has a single box adjacent to it across each of its six faces. In case of grid refinement, there are two other cases. If the adjacent box is refined, the density used for upwinding is obtained by averaging the densities for the four adjacent refined elements. If the adjacent box is coarser, then three densities are averaged. In two dimensions, the possibilities for upwinding to the left across an edge are illustrated in Figure 2.6.

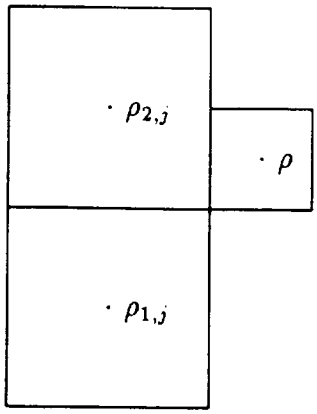
Upwinded density $\tilde{\rho}$ is defined by

$$\tilde{\rho} = \rho + \mu \sum_{i=1}^{6} max(-\hat{V} \cdot \vec{n}_i, 0) S_i(\vec{V}) \sum_j C_{i,j}(\rho_{i,j} - \rho) \tag{2.43}$$

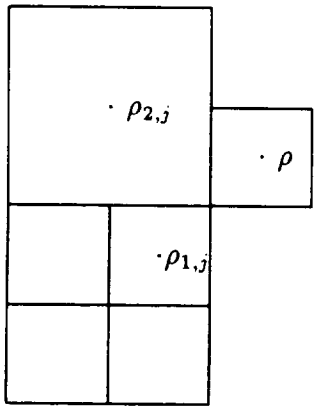
where i runs over the 6 faces of the given box, j runs over the densities averaged to obtain the density upwinded to, $C_{i,j}$ is the coefficient for each of the four densities contributing to the density upwinded to, $\hat{V}$ is the normalized velocity at the centroid of the given element, $\vec{n}_i$ is the outward pointing normal to face i of the element, and $S_i(\vec{V})$ is a cubic blending function to make the upwinding differentiable. This upwinding is first order accurate, introducing an error comparable to replacing the density ρ with a piecewise constant approximation in each element. In the case of D-regions, special operators must be constructed based on local information about box adjacency. This information is extracted from the oct-tree (see Appendix A) and D-region lists in a preprocessing step.



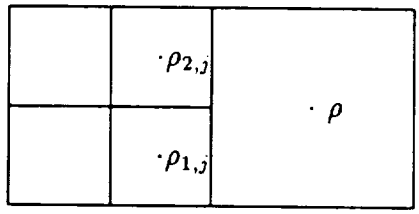
Face Adjacent Box at Same Level



Face Adjacent Box is Less Refined



Face Adjacent Box is Less Refined



Face Adjacent Box is More Refined

Figure 2.6: Upwinding Stencils in Two Dimensions for Negative x Edge.

2.3.8 Accuracy of Discretization

By keeping the trial functions parameterized by values at the corners of similar boxes, uniformity of the basis [58] is guaranteed. Thus, in the limit, standard approximation theory and finite element error estimates hold. The asymptotic convergence of the method has been verified with uniform grids for the case of a sphere in incompressible flow where an analytic solution is available [36]. Sections 3.1 and 3.2 contain computational examples that demonstrate the method's accuracy for locally refined grids. There is no theoretical guarantee of good conditioning. However, experience to date with the code has not uncovered any conditioning problems for the cases in the range from 8000 to 600000 grid points and refined grids with relative levels of 10 below the global grid level.

2.4 SOLUTION ALGORITHM

2.4.1 Linear Solution Algorithm

The solution technique used in TRANAIR was designed for nonlinear problems. However, it is useful to describe the algorithm applied to the special case of a linear boundary value problem.

Discrete System

The generic linear potential equation

$$\vec{\nabla} \cdot (\rho \vec{\nabla} \Phi) = f \tag{2.44}$$

is considered with $\rho = \rho(x, y, z)$ assumed to be given and strictly positive. The boundary conditions are those described earlier in Section 2.1.

In order to enforce the far field condition given by Eqn. (2.3), source unknowns Q are introduced on the global grid and replace unknowns Φ there. Since Q is known to be zero on the boundary of the global grid, the residual does not need to be computed there. The extrapolated values in boundary boxes are denoted by Ψ, all other variables on the refined grid are denoted by Φ, and the doublet parameters on wakes are denoted by μ. The finite element operator described in Section 2.3.4 will be denoted by L. It is defined over the whole grid except on the boundary of the global grid and is evaluated by multiplying the element stiffness matrices by the vector of unknowns.

Thus it is necessary to solve the linear system of equations

$$L \begin{pmatrix} T^{-1}Q \\ \Phi \\ \Psi \\ \mu \end{pmatrix} = f. \tag{2.45}$$

Preconditioned System

Since the system (2.45) (depending on boundary conditions) is non-symmetric and non-definite the GMRES method of Saad and Schultz [53] is chosen as the basic iterative solver.

The operator, T^{-1}, used to obtain the potential from the sources acts as an effective right preconditioner for the global grid points. It is also necessary to use a left preconditioner, to approximate the problem near the internal boundaries. The left preconditioner, N, is taken to be the global stiffness matrix restricted to a *reduced set* of unknowns. The reduced set is defined to consist of all unknowns located at corners of boundary boxes, refined boxes, or boxes with total pressure or temperature different form free stream values, and the doublet parameters μ. The stagnation unknowns (those that are located in the interior of the configuration) are not included in the reduced set. The reduced set is closed by *closure unknowns* which are outside the reduced set but in the stiffness matrix stencil of some unknown in the reduced set.

The boundary condition at closure unknowns is an approximation to the far field condition for the original problem, i.e., $\phi = 0$.

Note that there is some overlap between the Q unknowns on the global grid preconditioned by T^{-1} and those in the reduced set preconditioned by N^{-1}. For unknowns Q at global grid points in the reduced set, an additional preconditioner T must be applied on the left to make the equation dimensionally correct.

Hence, in all there are five classes of unknowns in a given flow problem. They are:

- $Q^{(1)}$, the source unknowns at global grid points which are not in the reduced set and not in stagnation regions;
- $Q^{(2)}$, the source unknowns at global grid points in the reduced set or in stagnation regions;
- Φ, the values of the velocity potential at points on locally refined grids;
- Ψ, the values of velocity potential in the boundary basis functions;
- μ, the doublet strengths at leading edges of wake networks.

The preconditioned equation can then be written as

$$TN^{-1}(f - LT^{-1}\mathcal{X}) = 0 \tag{2.46}$$

where

$$\mathcal{X} = \begin{pmatrix} Q^{(1)} \\ Q^{(2)} \\ \Phi \\ \Psi \\ \mu \end{pmatrix}. \tag{2.47}$$

The operators T and N are defined as:

$$T = \begin{pmatrix} T_{(1)(1)} & T_{(1)(2)} & 0 & 0 & 0 \\ T_{(2)(1)} & T_{(2)(2)} & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{pmatrix} \tag{2.48}$$

$$N = \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ 0 & N_{(2)(2)} & N_{(2)(3)} & N_{(2)(4)} & N_{(2)(5)} \\ 0 & N_{(3)(2)} & N_{(3)(3)} & N_{(3)(4)} & N_{(3)(5)} \\ 0 & N_{(4)(2)} & N_{(4)(3)} & N_{(4)(4)} & N_{(4)(5)} \\ 0 & N_{(5)(2)} & N_{(5)(3)} & N_{(5)(4)} & N_{(5)(5)} \end{pmatrix}. \tag{2.49}$$

To achieve invariance with respect to units, the source unknowns Q must be scaled relative to the potential unknowns (the scale factor has the dimension of the inverse length squared). After scaling, the GMRES convergence history is independent of the physical units used to define the problem.

Preconditioned Residual

The calculation of the preconditioned residual R (the function evaluation subroutine for GMRES) is now described.

For unknowns $Q^{(1)}$ (those on the global grid but not in the interior of the reduced set)

$$R(Q^{(1)}) = f - L \begin{pmatrix} T^{-1} \begin{pmatrix} Q^{(1)} \\ Q^{(2)} \end{pmatrix} \\ \Phi \\ \Psi \\ \mu \end{pmatrix}. \tag{2.50}$$

For unknowns $Q^{(2)}$ (those on the global grid and in the interior of the reduced set or located at global grid points in stagnation regions)

$$R(Q^{(2)}) = TN^{-1} \left(f - L \begin{pmatrix} T^{-1} \begin{pmatrix} Q^{(1)} \\ Q^{(2)} \end{pmatrix} \\ \Phi \\ \Psi \\ \mu \end{pmatrix} \right). \tag{2.51}$$

In Eqn. (2.51), special account must be taken of unknowns Q located in stagnation regions, such as the interior of wings and fuselages. For these unknowns, it is important to realize that N^{-1} is just the identity, $f = 0$, and $L\Phi = \phi$. Thus T is applied to the global grid unknowns in the reduced set, closure point unknowns, and stagnation unknowns. But the input for T at stagnation unknowns comes from a different process than that used for the other two classes of Q unknowns. Another special class of Q unknowns in Eqn. (2.51) are those at closure points. N^{-1} does apply to the residual at these points producing input values for T to give residual values for points in the reduced set. But for these closure unknowns, the residual is actually given by Eqn. (2.50).

For unknowns Φ not located on the global grid and for all unknowns Ψ and μ

$$\begin{pmatrix} R(\Phi) \\ R(\Psi) \\ R(\mu) \end{pmatrix} = N^{-1}L \begin{pmatrix} T^{-1} \begin{pmatrix} Q^{(1)} \\ Q^{(2)} \end{pmatrix} \\ \Phi \\ \Psi \\ \mu \end{pmatrix}. \tag{2.52}$$

For Φ unknowns located at points not in the global grid but in stagnation regions, the residual is given by $R(\Phi) = L\Phi = \phi$.

Preconditioners

The operator T^{-1} represents the discrete Green's function and is defined over the uniform global grid. Construction and application of the discrete Green's function (Poisson Solver) is extremely fast since one can take advantage of the constant grid

spacing and use discrete Fourier transforms. More details on this preconditioner are give in Appendix D.

The left preconditioner matrix N is sparse and it is feasible to do a direct sparse incomplete factorization of N. This works for the following reasons. First, a drop tolerance can be introduced into the sparse elimination process allowing small elements in the decomposition to be dropped as they are generated. This has a cascading effect and reduces fill dramatically [43]. Second, a grid based nested dissection ordering can be generated which reduces fill during elimination and therefore the total amount of work. In most cases the drop tolerance is the most effective strategy. Figure 2.7 shows the reduced set and a possible first dissector for a grid for a sphere case. More details on the sparse solver preconditioner are given in Appendix E.

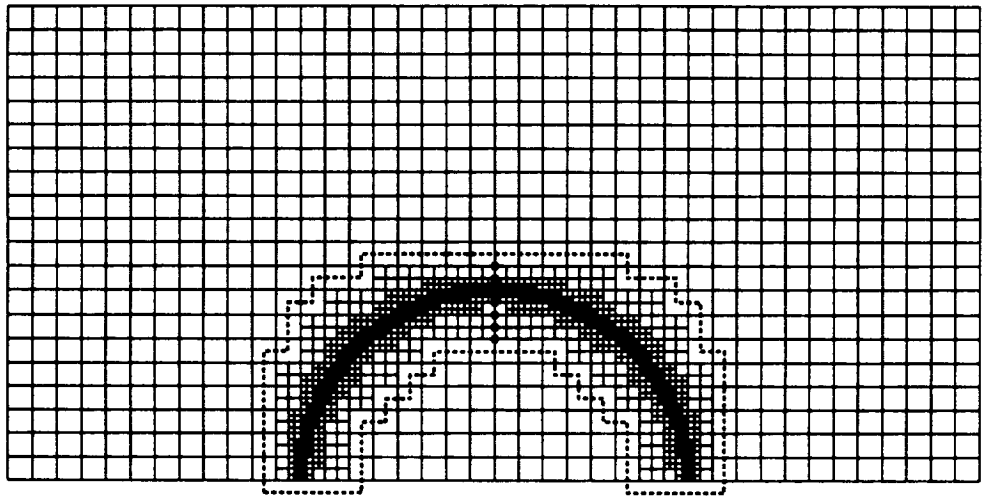


Figure 2.7: Reduced Set and Possible First Dissector.

2.4.2 Nonlinear Solution Algorithm

When the problem is nonlinear it is necessary to use the Newton method. Each step of the Newton method requires the solution of a linear problem of the type discussed in Section 2.4.1.

Newton Method

Consider the nonlinear system of equations

$$F(x) = 0. \tag{2.53}$$

Given an initial approximate solution x^0, for $n = 0, 1, 2, \ldots$ until the residual is sufficiently small, set

$$x^{n+1} = x^n + \lambda(\delta x^{n+1}) \tag{2.54}$$

where δx^{n+1} is the solution of the linear system

$$\overline{F}_{x^n}(\delta x^{n+1}) = -F(x^n) \tag{2.55}$$

and λ is a step length to be determined. Here $\overline{F}_{x^n}$ is the Jacobian for F linearized about x^n. This linear operator can be defined by giving its action on any vector y

$$\overline{F}_x(y) = \epsilon \overset{lim}{\rightarrow} 0 \frac{F(x+\epsilon y) - F(x)}{\epsilon}. \tag{2.56}$$

The step length λ is selected so that

$$\|F(x^{n+1})\| < \|F(x^n)\| \tag{2.57}$$

in some appropriate norm.

The GMRES algorithm can be used to solve Eqn. (2.55). This algorithm requires only the ability to calculate the action of the linear operator $\overline{F}_x$ on any vector y. Equation (2.56) can be used to approximate this action

$$\overline{F}_x(y) \simeq \frac{F(x+\epsilon y) - F(x)}{\epsilon} \tag{2.58}$$

where ϵ is small in some appropriate sense. Thus, the linear problem, Eqn. (2.55), can be solved without ever explicitly generating the Jacobian for the full nonlinear problem.

To control the cost of the method Eqn. (2.55) is solved only approximately with GMRES, i.e., δx^{n+1} satisfies

$$\|\overline{F}_{x^n}(\delta x^{n+1}) + F(x^n)\| < \eta.$$

This makes the method an inexact Newton method [59]. If η is constant, the method converges linearly. If η goes to 0 as convergence takes place, the convergence is superlinear. More details can be found in [54].

Preconditioning GMRES

Preconditioning Eqn. (2.55) is identical to that used for linear systems and given in Eqn. (2.46). If f is replaced by $-F(x^n)$, L by $\overline{F}_{x^n}$, and $T^{-1}\mathcal{X}$ by x, Eqn. (2.46) is the same as Eqn. (2.55) preconditioned on the left by TN^{-1}. For convenience, the finite difference formula (2.58) is applied to $TN^{-1}F$ rather than F. The matrix forms of T, N, and T^{-1} are given in Section 2.4.1.

The matrix N is an approximation to the Jacobian for F about the current solution restricted to a reduced set as described in Section 2.4.1 above. The reduced set now also includes all elements where upwinding is used.

The matrix N is generated on an element by element basis using the element stiffness matrices. The density function ρ and its derivatives are evaluated at element centroids. For unknowns one of whose eight contributing elements has upwinding in effect, the row of the matrix N depends on more than just these eight elements. The algorithm can be simplified by applying the chain rule to the calculation of a matrix entry,

$$\frac{dF(\Phi)_i}{d\Phi_j} = \frac{\partial F(\Phi)_i}{\partial \Phi_j} + \sum_k \frac{\partial F(\Phi)_i}{\partial \tilde{\rho}_k} \cdot \frac{\partial \tilde{\rho}_k}{\partial \Phi_j}. \tag{2.59}$$

where $\tilde{\rho}$ is given by Eqn. (2.39). The first term on the right is the contribution from the subsonic stencil, i.e., the element stiffness matrices. The second term is generated using a sparse matrix-matrix multiply. This technique enables vectorization even though the upwinding is element dependent.

The convergence of the inexact Newton method depends on how well the matrix N represents the Jacobian of F. If the damping strategies described below are used it is usually necessary to compute and invert matrix N infrequently.

Local Damping of Newton Method

Newton's method is rarely globally convergent. Also, its convergence rate is generally quadratic only sufficiently close to the solution. The initial iterate is usually taken to be $\phi = 0$, which is not a good approximation to the solution. Thus, Newton's method works well only for moderate to small problems or those with only weak or no shocks. For large problems or problems with reasonably strong shocks, Newton's method must be damped to prevent divergence or very slow convergence.

Various damping strategies have been tested in the present method. One due to Bank and Rose [60] for determining the step length λ is based on the residual for Eqn. (2.1). This strategy is fairly simple to implement and provides adequate local damping in many cases.

Another strategy is to limit λ to prevent local Mach numbers greater than some prescribed cut off value. This prevents spurious large velocities from causing stagnation of convergence. In the ONERA M6 wing results reported below, this strategy was used with a local Mach number cutoff of $\sqrt{5}$.

However, local damping strategies of this kind are only effective by themselves in cases that almost converge anyway. In more difficult transonic cases, a steep shock can form in the wrong location early in the iterative process and the Newton method can stagnate. In this case, a local method can rarely move the shock more than one grid point per iteration, resulting in very slow convergence. This situation seems to be due to the fact that the residual is much larger near the shock than elsewhere.

The shortcomings of the local damping strategies can be seen in the case of the ONERA M6 wing at $M_\infty = 0.84$ and angle of attack $\alpha = 3.06°$ on a grid having about 311,000 elements. This case exhibits a strong shock outboard as well as an oblique

shock. If Newton's method is used with an initial iterate $\phi = 0$ and the Bank-Rose strategy for limiting λ, the convergence stagnates at the iterate shown in Figure 2.8. The final converged solution is shown for reference. If λ is further limited to control local Mach numbers as described above, the convergence is still very slow. Figure 2.9 shows the Newton iterate after six and twelve Newton steps. Newton's method is moving the shock toward the correct location very slowly.

Viscosity Damping

To improve convergence in the presence of shock waves, a problem dependent dissipation is used. Here a larger amount of dissipation is introduced during the early iterations and it is reduced to appropriate levels as the solution develops. This type of damping strategy can be implemented through a continuation process which can be based on many types of parameters.

In the first, and more direct approach (called the *viscosity damping strategy*), the discrete problem is modified by multiplying the switching function of Eqn. (2.54) by a constant factor (1.5 to 3.0) and by reducing the cut-off Mach number during the initial steps in the Newton method. This has the effect of increasing the amount of artificial viscosity and applying it to a larger part of the flow field. After several Newton steps, the problem is modified by reducing the multiplying factor and raising the cut-off Mach number. This process is repeated until the desired level of dissipation is reached. This continuation process works very well since it has the effect of locating the supersonic zone and the shock position fairly early in the process, even though the shock is quite smeared.

When, viscosity damping is used in the case of the ONERA M6 wing, convergence improves considerably after the initial viscous problems are partially solved. A partially converged solution at the second continuation step (Newton step seven) is shown in Figure 2.10. Figure 2.11 shows the convergence histories for these runs. The residual jumps in this figure correspond to discrete changes in the continuation parameter. The drawback of this continuation approach is the high cost of even partially solving the viscous problems that are introduced.

Several other parameters were used as continuation parameters including free stream Mach number (M_∞) and the total pressure of the free stream. In both cases, the shock location was sensitive to the continuation parameter and convergence was poor in certain cases.

2.4.3 Grid Sequencing

A strategy that has proven to be very reliable for ensuring convergence for difficult transonic problems is grid sequencing. Basically the process involves the following steps. A sequence of coarse to fine grids are generated *a priori*. The solution is found on the coarsest grid. The converged solution is interpolated onto the next finer grid and the problem is solved on that grid. This is repeated until the solution is obtained on the finest grid. A gradual change in viscosity is brought about by the fact that the grid cell size in the initial stages (on coarse grids) is larger and thus the dissipation

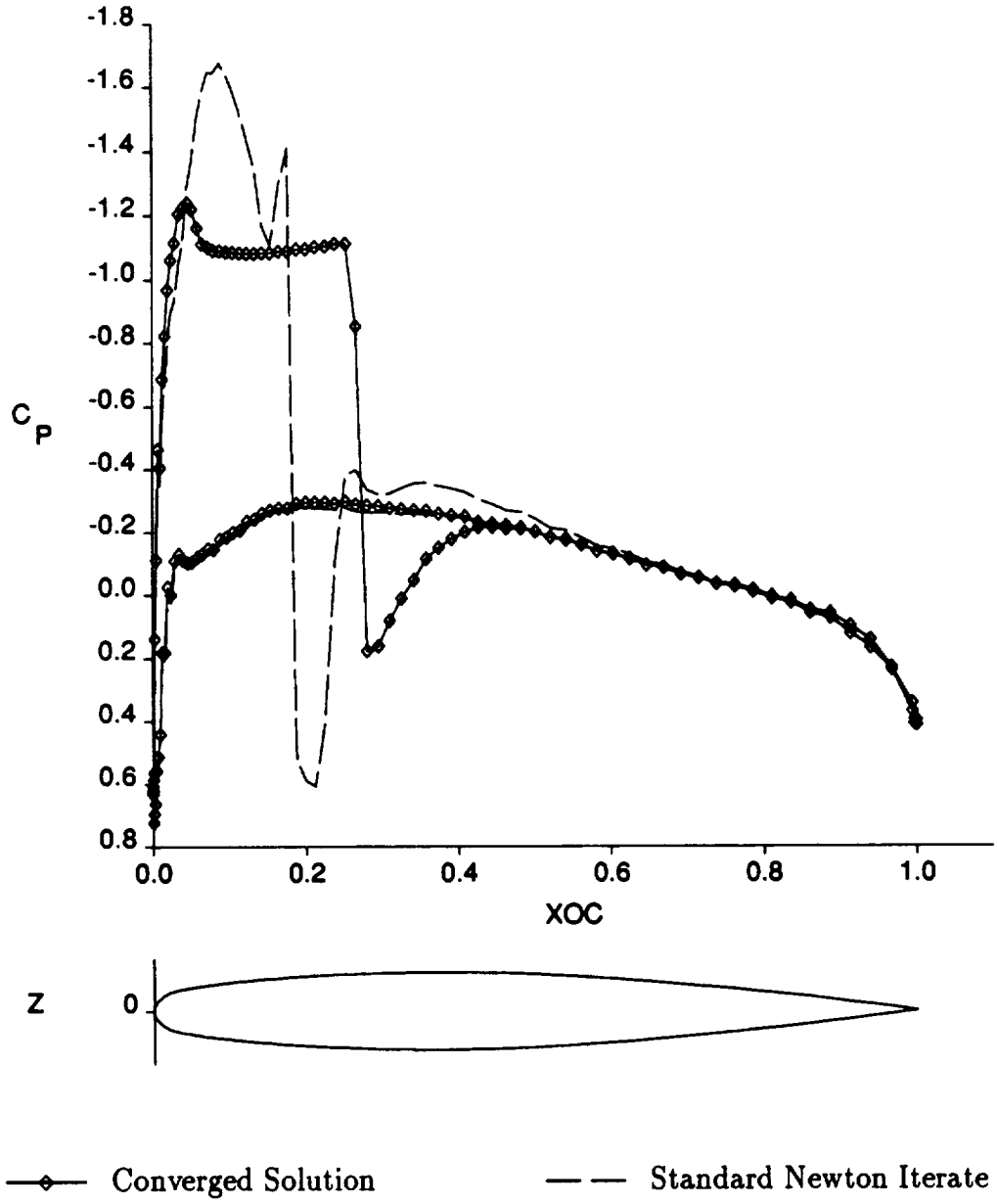


Figure 2.8: Iterate for Newton's Method With Residual Damping for ONERA M6 Wing Case, $M_\infty = 0.84, \alpha = 3.06°$, 91% Span Station.

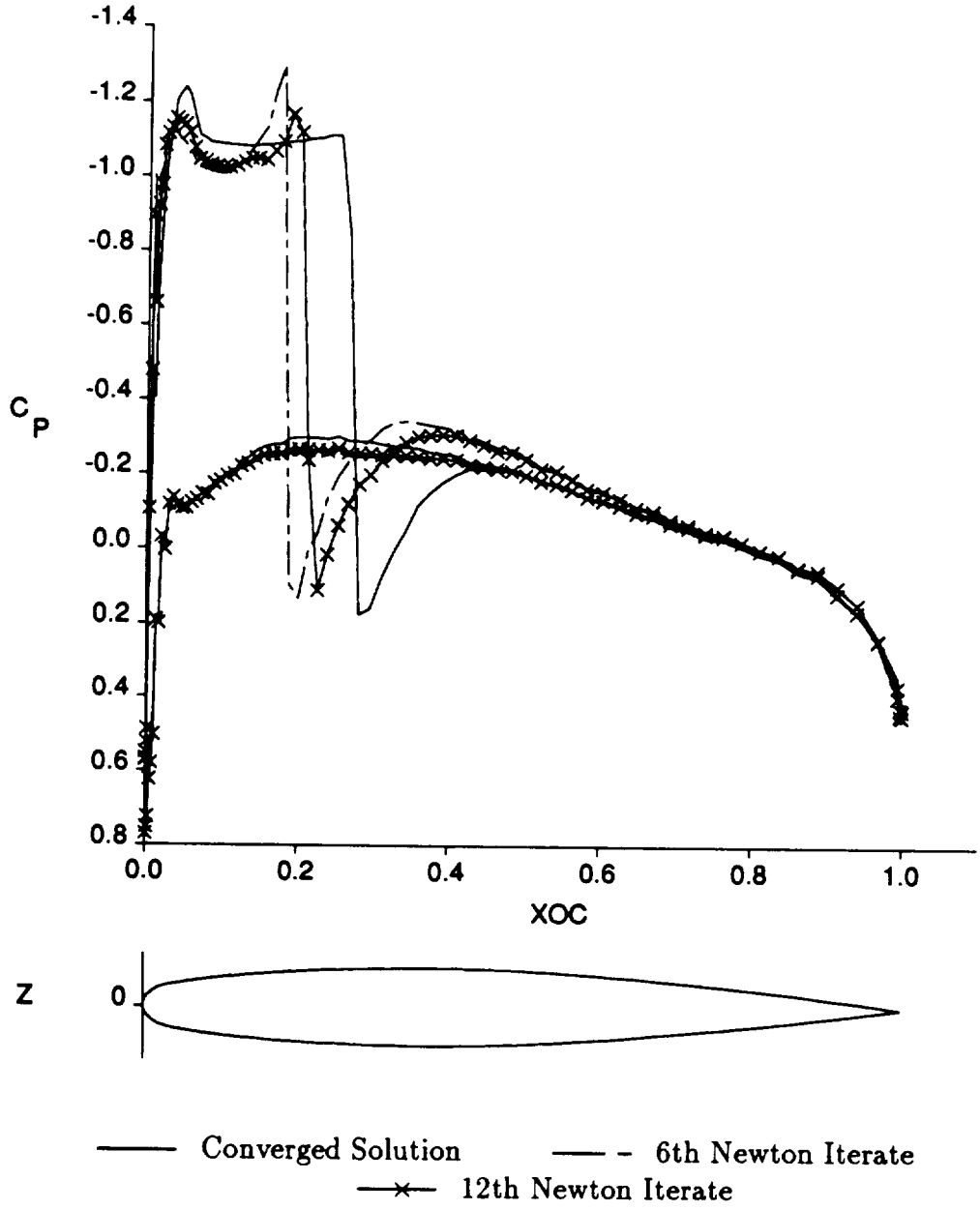


Figure 2.9: Iterates for Newton's Method With Residual and Local Mach Number Damping for ONERA M6 Wing Case, $M_\infty = 0.84, \alpha = 3.06°$, 91% Span Station.

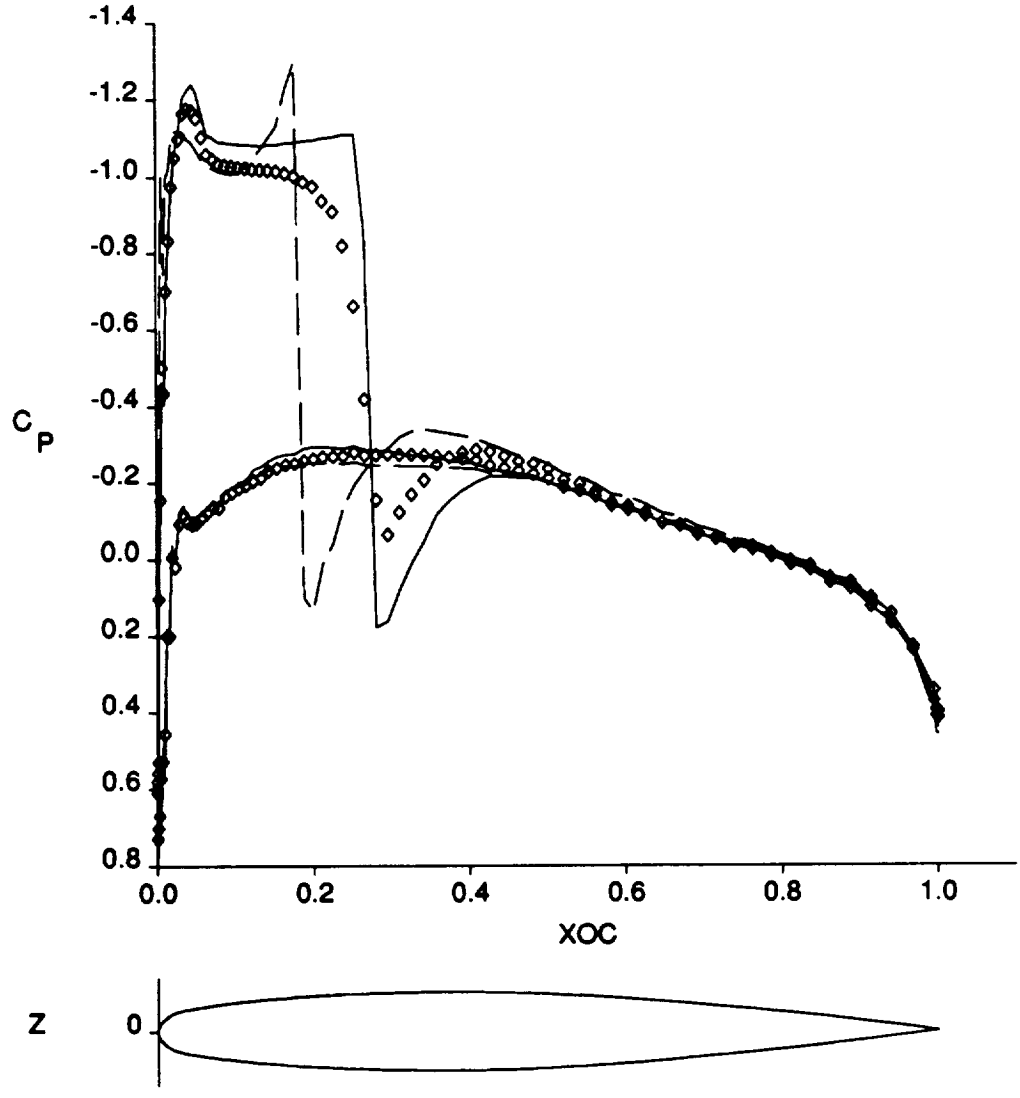
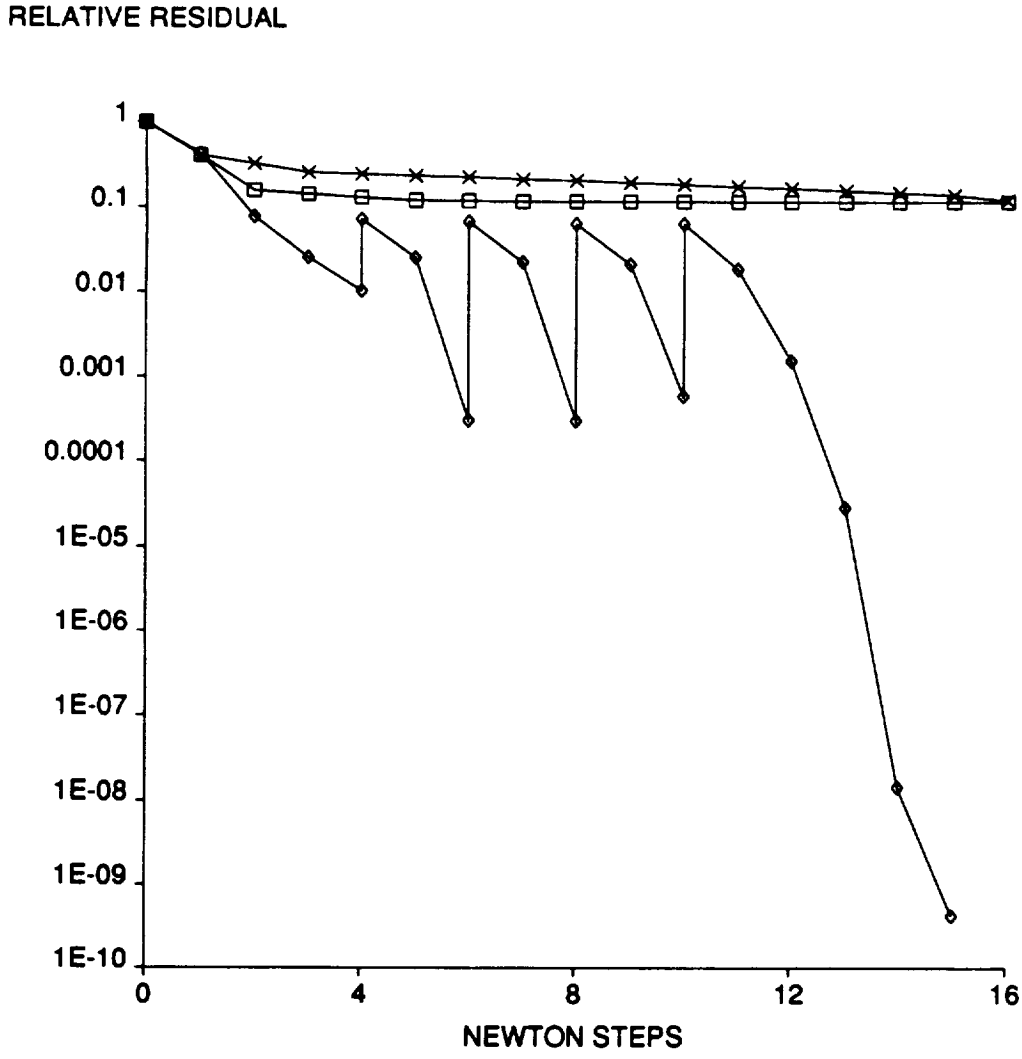


Figure 2.10: Partially Converged Iterate for the Second Continuation Step Using Viscosity Damping for ONERA M6 Wing Case, $M_\infty = 0.84, \alpha = 3.06°$, 91% Span Station.



—□— Residual Damping
—×— Residual and Local Mach Number Damping
—◇— Residual, Local Mach Number, and Viscosity Damping

Figure 2.11: Convergence Histories for Newton's Method with Various Damping Strategies for ONERA M6 Wing Case, $M_\infty = 0.84, \alpha = 3.06°$.

is larger. As the grid becomes finer the dissipation is automatically reduced. The process of interpolating the solution naturally positions the nonlinear features in the solution. It is also possible to employ viscosity damping on any or all the grids in the sequence with some simple code modifications. The benefits of this approach are more reliable convergence and lower computer cost.

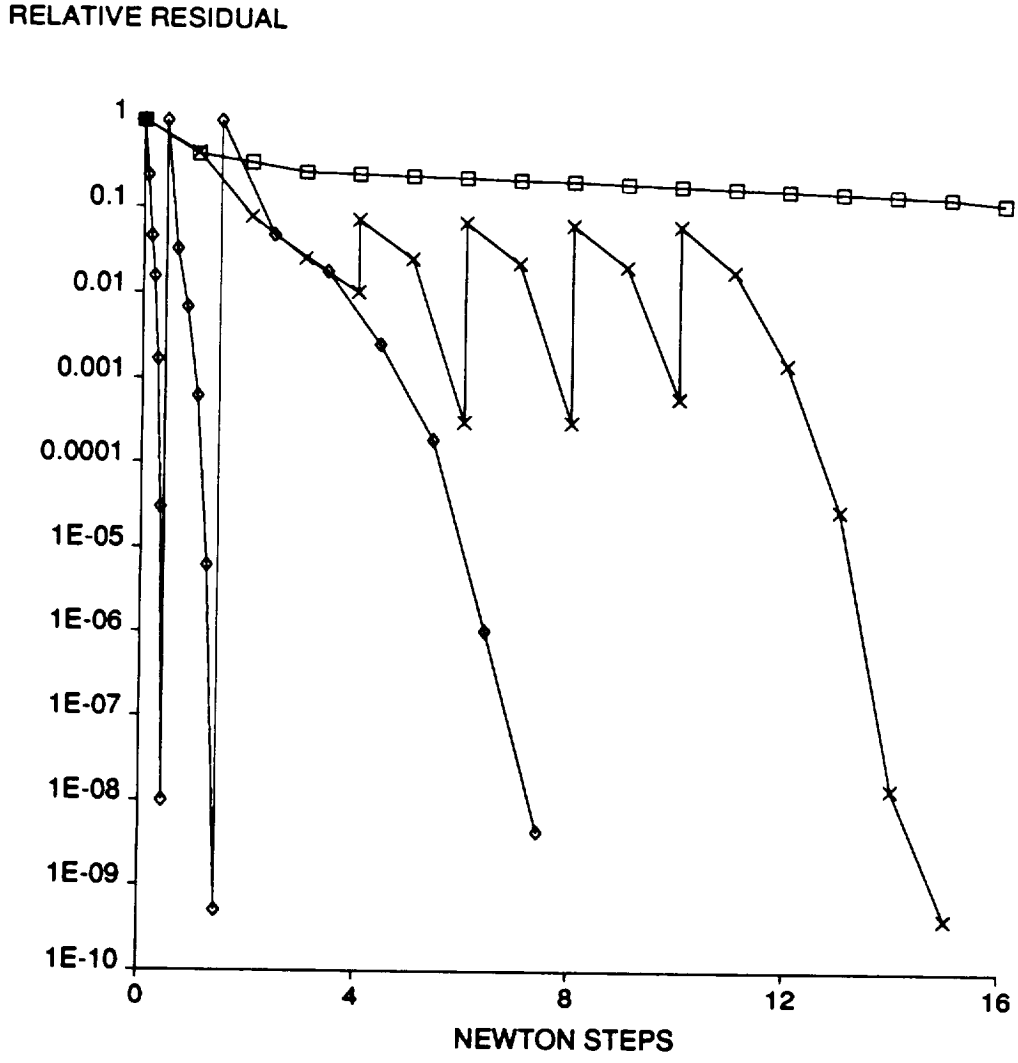
The results for the ONERA M6 case are presented below. With grid sequencing, this case converged rapidly. As discussed in Section 2.4.1, this case did not converge when residual and local Mach number damping was used with Newton's method with an initial guess of $\phi = 0$. Convergence was obtained in two ways. Initially, viscosity damping was used and it was found that four continuation steps were required. With grid sequencing, this case converged more rapidly and CPU times were proportionally reduced. Figure 2.12 compares convergence histories for these three methods. Iterations in the grid sequencing run are scaled by the approximate size of the problem for the early small grids (this scaling corresponds approximately to CPU cost). Grid sequencing offers a substantial advantage in both rate of convergence and storage requirements. For the grid sequencing run, CPU time was about half of that needed for the viscosity damping run.

Cuts through the three grids used are shown in Figures 2.13 through 2.16. The final fine grid is the last of these three grids. The grids had about 19,000, 56,000, and 311,000 elements respectively. Figure 2.17 shows surface pressures obtained on the three grids used in this case. On the coarser grids, the shock is in the right location but smeared.

2.4.4 Solution Adaptive Grids

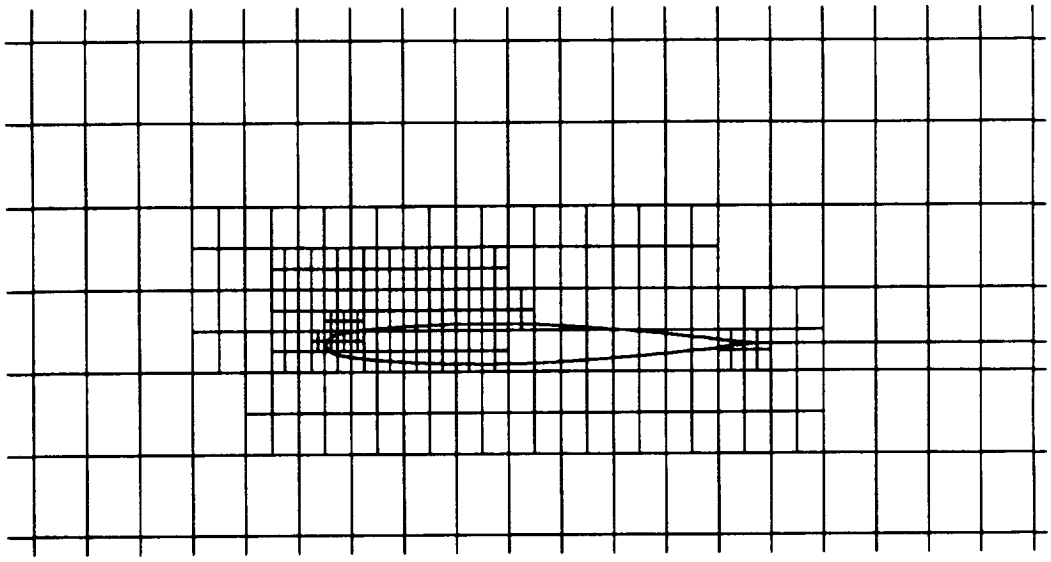
The grid sequencing method described in Section 2.4.3 operates on a set of preconstructed grids in which the refinement is governed *a priori* by the configuration surface definition and other specifications (see Section 2.3.3). The solution adaptive grid method starts with the coarsest grid in the preconstructed sequence of grids. The overall procedure in discretizing the problem on the current grid and solving the discrete equations is identical to that described so far (see Sections 2.3 and 2.4). However the solution adaptive method differs from the grid sequencing method in two regards. The first difference is that in the adaptive method, the next grid is generated anew and the local resolution of the new grid is determined by *a posteriori* computed local error estimates and user inputs (rather than being taken from the pre determined sequence). The second difference is that in creating the new grid single-level local refinement *as well as* derefinement may be used. In refining, a rectangular box element is replaced by eight smaller similar elements, whereas, in derefining, eight sibling elements are coalesced to form a larger similar element.

The goal of the adaptive grid method is to obtain a final grid with a (specified) target number of elements, N, and a numerical solution on that grid that is nearly as accurate as the best solution one could obtain using N elements in any grid. To achieve this goal five sequential steps are carried out in creating each new grid. These steps consist of estimating local errors on the previous grid, computing local error predictors, applying *a priori* grid refinement controls, applying a grid refinement

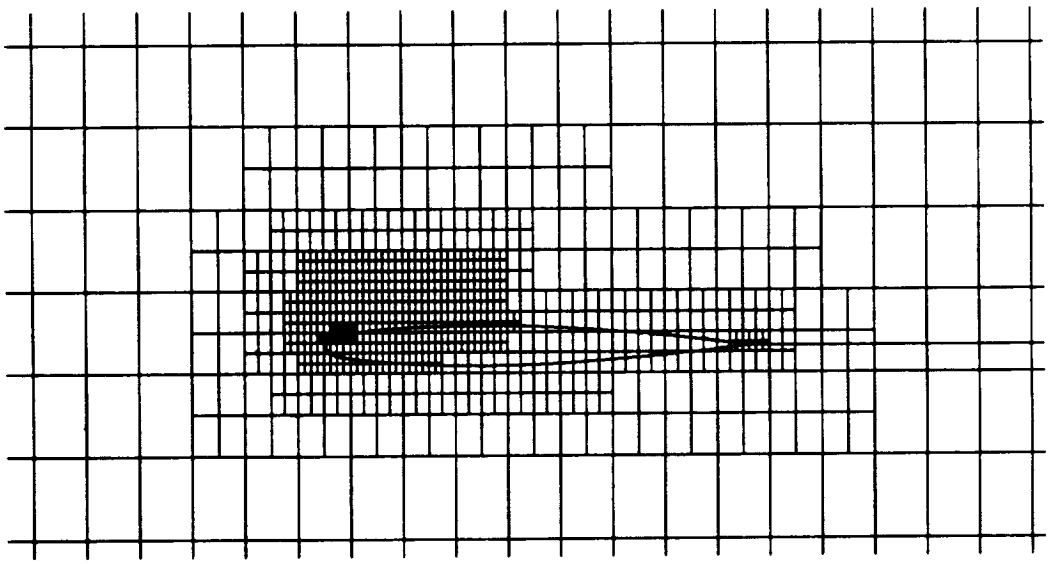


Standard Newton Damping
Viscosity Damping
Grid Sequencing

Figure 2.12: Convergence Histories for Newton's Method, Newton's Method with Viscosity Damping, and Grid Sequencing for ONERA M6 Wing Case, $M_\infty = 0.84, \alpha = 3.06°$.

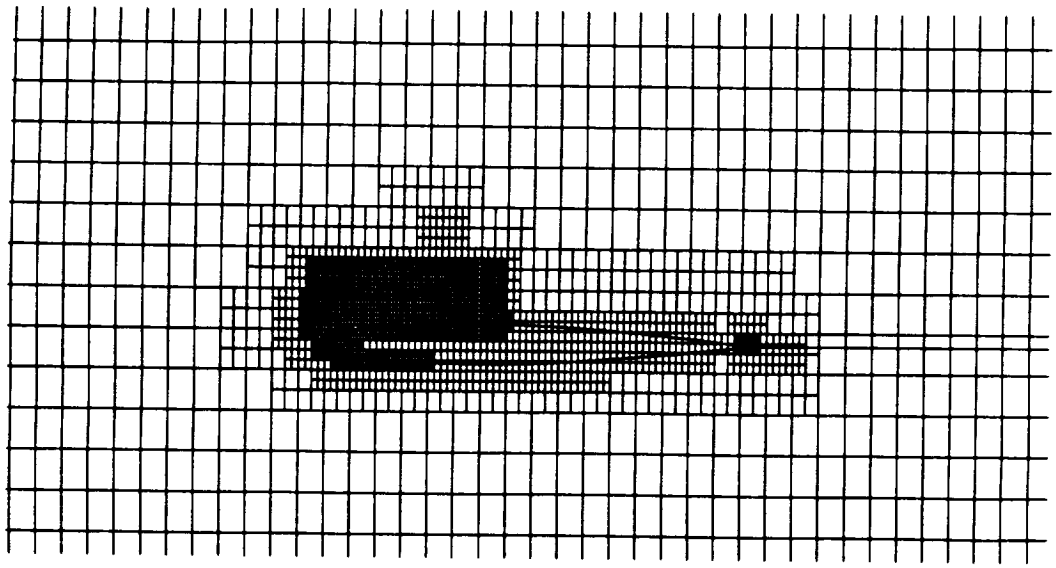


Coarse Grid



Medium Grid

Figure 2.13: Cuts Through The Coarse and Medium Grids Generated by Grid Sequencing for ONERA M6 Wing at 91% Span, $M_\infty = 0.84, \alpha = 3.06°$.



Fine Grid

Figure 2.14: Cut Through the Fine Grid Generated by Grid Sequencing for ONERA M6 Wing at 91% Span, $M_\infty = 0.84, \alpha = 3.06°$.

strategy, and constructing the new grid.

Use of the method components in these steps is novel in the present context, but the basic ideas behind them have been proposed before by other researchers [61]-[68]. The performance of the present solution adaptive grid method depends somewhat strongly on the characteristics of individual applications, and the specific method components employed. The specific method components were chosen after substantial but nonexhaustive, testing and analysis.

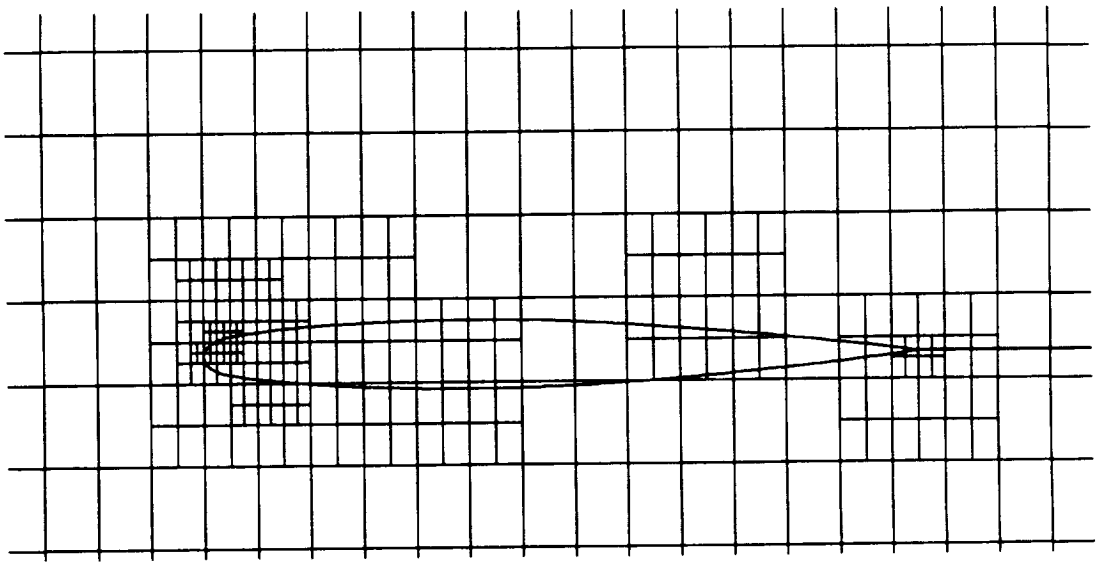
Computing Local Error Estimates

Local differences of velocity components are used as estimates of the error for each rectangular[1] element in the grid. The error estimate for an element consists of

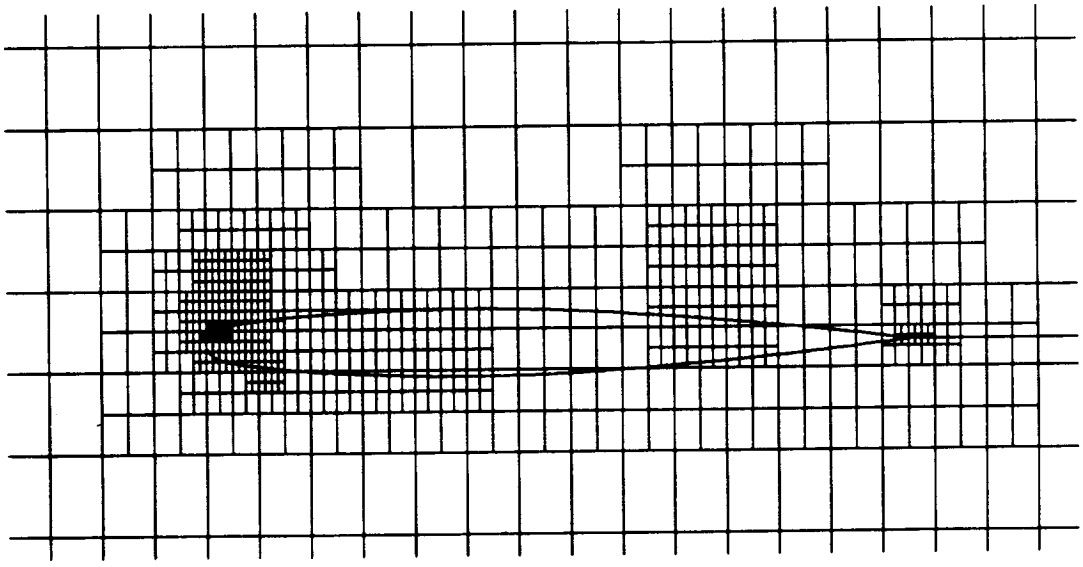
$$errest \equiv \max_{region\ r} \{\max_{j} \{(\Delta v_1^{r,j})^2 + (\Delta v_2^{r,j})^2 + (\Delta v_3^{r,j})^2\}^{\frac{1}{2}}\}, \tag{2.60}$$

where, for the rth solution region contained in the element, $\Delta v_i^{r,j}$ denotes the difference across the element's jth face of the region centroid values of the ith velocity component. The outer maximum in Eqn. (2.60) is taken over all regions contained

[1]It should be recognized that the finite element method is applied on the regions over which the element trial functions are defined. These regions are part of the rectangular box elements. The grid refinement process refines box elements, and new regions are determined for the subdivided box elements. The grid refinement process does not refine individual regions.

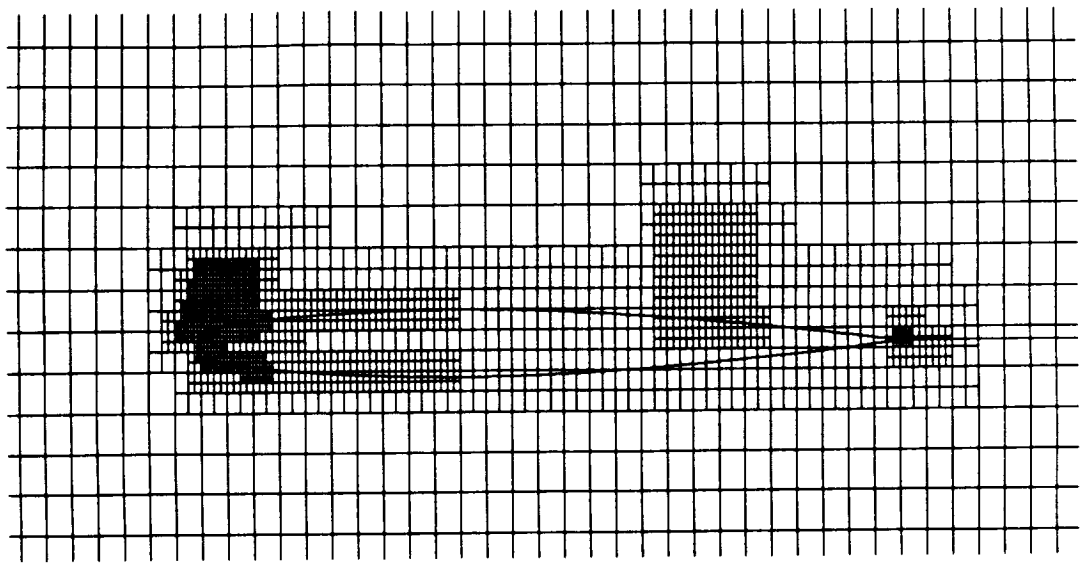


Coarse Grid



Medium Grid

Figure 2.15: Cuts Through the Coarse and Medium Grids Generated by Grid Sequencing for ONERA M6 Wing at the Plane of Symmetry, $M_\infty = 0.84, \alpha = 3.06°$.



Fine Grid

Figure 2.16: Cut Through the Fine Grid Generated by Grid Sequencing for ONERA M6 Wing at the Plane of Symmetry, $M_\infty = 0.84, \alpha = 3.06°$.

in the element. The inner maximum is taken over all element faces connecting to a region not contained in a larger element. Figure 2.18 illustrates in the case of a two dimensional airfoil the directions in which velocity components are differenced to compute error estimates for five elements, labeled A–E, each of which contains only one solution region.

This error measure provides a natural way to detect flow features having different length scales near different configuration components and does not lead to excessive grid in the far field.

Computing Local Error Predictors

A simple local smoothing algorithm is used to form error predictors from the local error estimates. In this algorithm, nodal values are first set equal to the largest of the error indicators for adjacent elements and then interpolated at element centroids to form the predictors. This algorithm implicitly predicts the need for grid refinement near elements where large errors have been detected and spreads the effect of estimated errors by one or two elements, thus preventing *holes* in subsequent grids.

Applying *A Priori* Grid Refinement Controls

For accurate and efficient analyses of complex configurations, it is important that one is permitted to communicate regions of greatest and least interest to a solution adaptive grid code. In many cases this is essential so that flow features of interest can

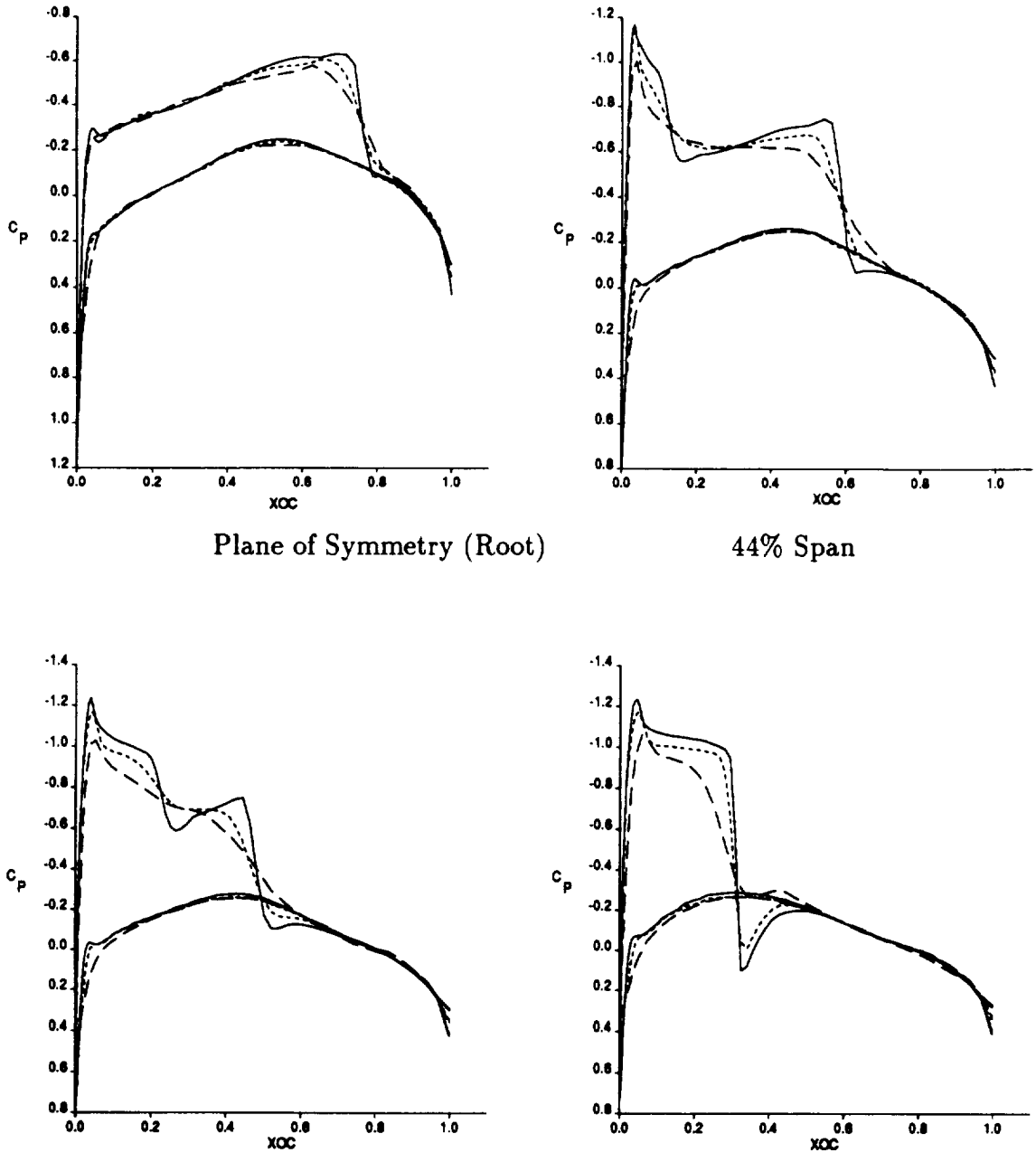


Figure 2.17: Surface Pressure for the Three Grids Generated by Grid Sequencing for ONERA M6 Wing, $M_\infty = 0.84$, $\alpha = 3.06°$.

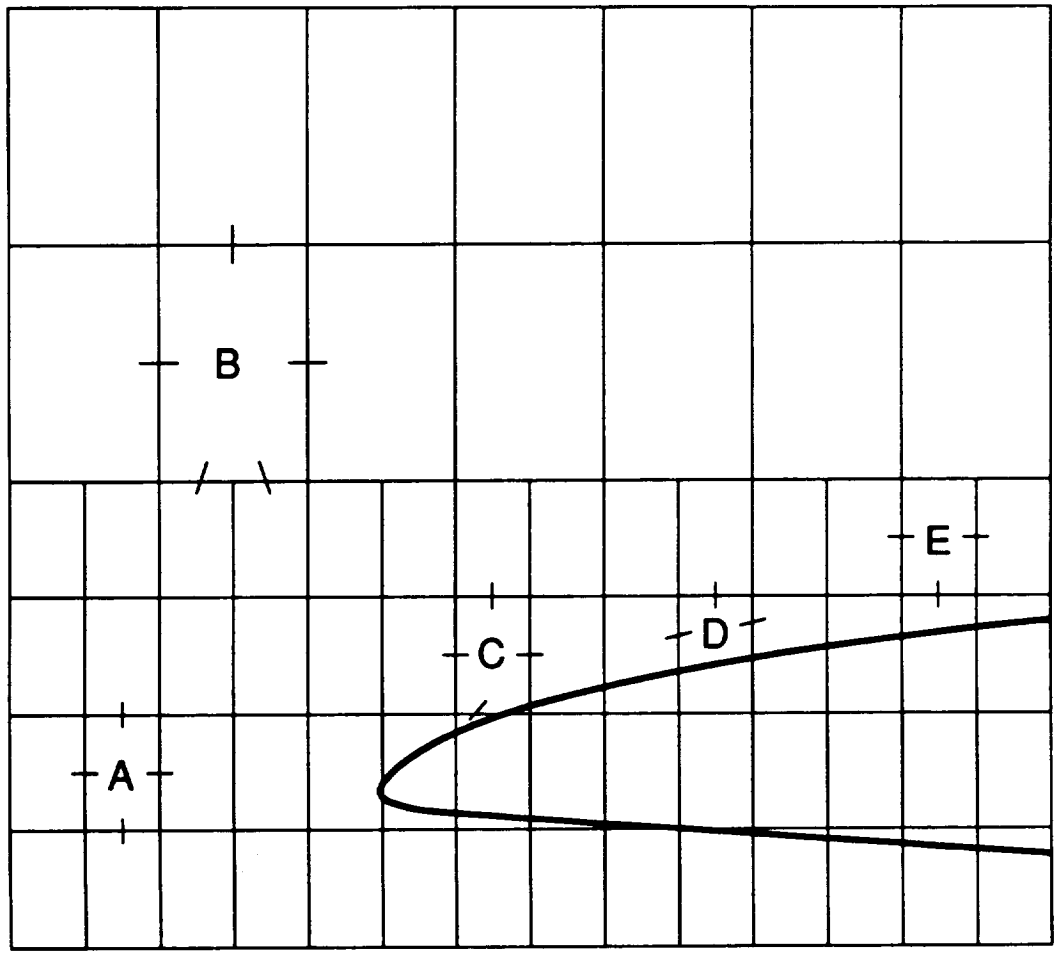


Figure 2.18: Directions for Velocity Component Differences in Error Indicators for Elements A–E

be resolved most accurately with a given target number of elements. The reasons for this are as follows. Unless otherwise instructed, the solution adaptive grid method gives equal weight to *all* regions with equal estimated errors in considering local grid refinement. Such regions could include those about wing tips, leading edges of many components in a configuration, irregularities in geometry, and wake regions, not all of which may be of equal importance to a person using the code. It can also happen that one flow feature that is easily detectable may dominate another flow feature that is latent (a feature that cannot be detected until sufficiently fine grid is present). For a given target number of elements, one often can significantly enhance the detection and resolution of latent flow features by restricting or de-emphasizing grid refinement in regions with dominant features.

The mechanism for exercising such *a priori* grid refinement controls in the solution adaptive grid method is the use of the same hexahedral shaped special boxes of interest (LBOs) used in constructing the initial fine grid (see Section 2.3). With each LBO, one specifies trilinearly varying minimum and maximum local grid sizes allowed in the LBO and a weight used to scale the corresponding local error predictors. The results of applying these controls in the method are an element refinement/derefinement eligibility list, a list of elements whose sizes are above specified maximum values, and a list of scaled error predictors ordered by size.

Applying Grid Refinement Strategy

In this step the elements are either marked for refinement, derefinement, or to be retained unchanged. Of the elements eligible for refinement, those with the largest scaled error predictors are marked for subdivision, with elements having grid sizes above specified maximum values taking precedence. Of the elements eligible for derefinement, those with the smallest scaled predictors are marked for derefinement (if all eight sibling elements are so eligible). The decisions regarding how many elements to refine and how many to derefine depend on a grid refinement strategy.

In examining strategies for deciding how many elements of each type to mark, two principles have proven useful. First, direct control should be exercised over the rates at which numbers of elements in successive grids increase. This excludes, for example, a prevalent strategy that refines/derefines solely on the basis of cut-off values which are proportional to the current mean local error indicator. Direct control is important because problem size can increase *very* rapidly with grid refinement in three dimensions. Second, for early and intermediate grids, grid refinement should be limited in regions where dominant flow features have been detected and be forced to occur in other regions. Failure to adhere to this principle can allow some flow features (e.g., leading edge expansions) to attract all available grid before other important features (e.g., shocks) develop.

A simple and flexible strategy following these principles is incorporated in the present method. It consists of

- refining and derefining fixed percentages of elements for most coarse and intermediate grids,

- attempting to more equally distribute local errors without significantly changing the number of elements in an intermediate grid, and
- only refining on the last grid.

More specifically (using a particular choice of input parameters for the method), with given intermediate and final target numbers of elements N_I and N_F, respectively, and N denoting the number of elements in the current grid such that $N_I < N_F < 4N_I$, and $N < N_I$

if $N < .4N_I$, 20% of the elements are marked for refinement and up to 40% for derefinement;

if $.4N_I < N < .9N_I$, only refinement is used to increase the number of elements to about N_I; and

if N is approximately equal to N_I, 2% of the elements are refined and up to 20% are derefined, and the next (final) grid adaptation consists of (only) refining enough elements so that the final grid has about N_F elements.

It is noted that the implementation of this and related grid refinement strategies consists of "solution adaptive grid cycles", where in each cycle, input consists of a target number of elements and various control parameters, and one or more adaptive grids are constructed. In the specific strategy described above, three cycles are used with target numbers of elements equal to N_I, N_I and N_F, respectively. In the second cycle only one adaptive grid is constructed.

Constructing a New Grid

Using a list of marked elements and a grid legalization constraint, a grid is constructed by building a new oct-tree structure. The legalization constraint, (see Appendix A), requires that additional elements be marked for refinement, if necessary, in order to prevent face-neighbor and edge-neighbor elements in the resulting grid from differing by more than one refinement level. In the newly constructed grid, the uniform global grid is expanded on the inflow or outer boundaries whenever a global grid box in the previous grid on the respective boundary is marked for refinement. Since linear flow assumptions are used on all inflow and outer boundary grid boxes, expansion of the grid (and problem domain) occurs whenever significant nonlinear effects are present in the flow near these boundaries.

Figure 2.19 illustrates the types of grids created in an application of the solution adaptive grid method. Pictured there are cuts of the initial grid and the second and fourth adaptive grids in a run.

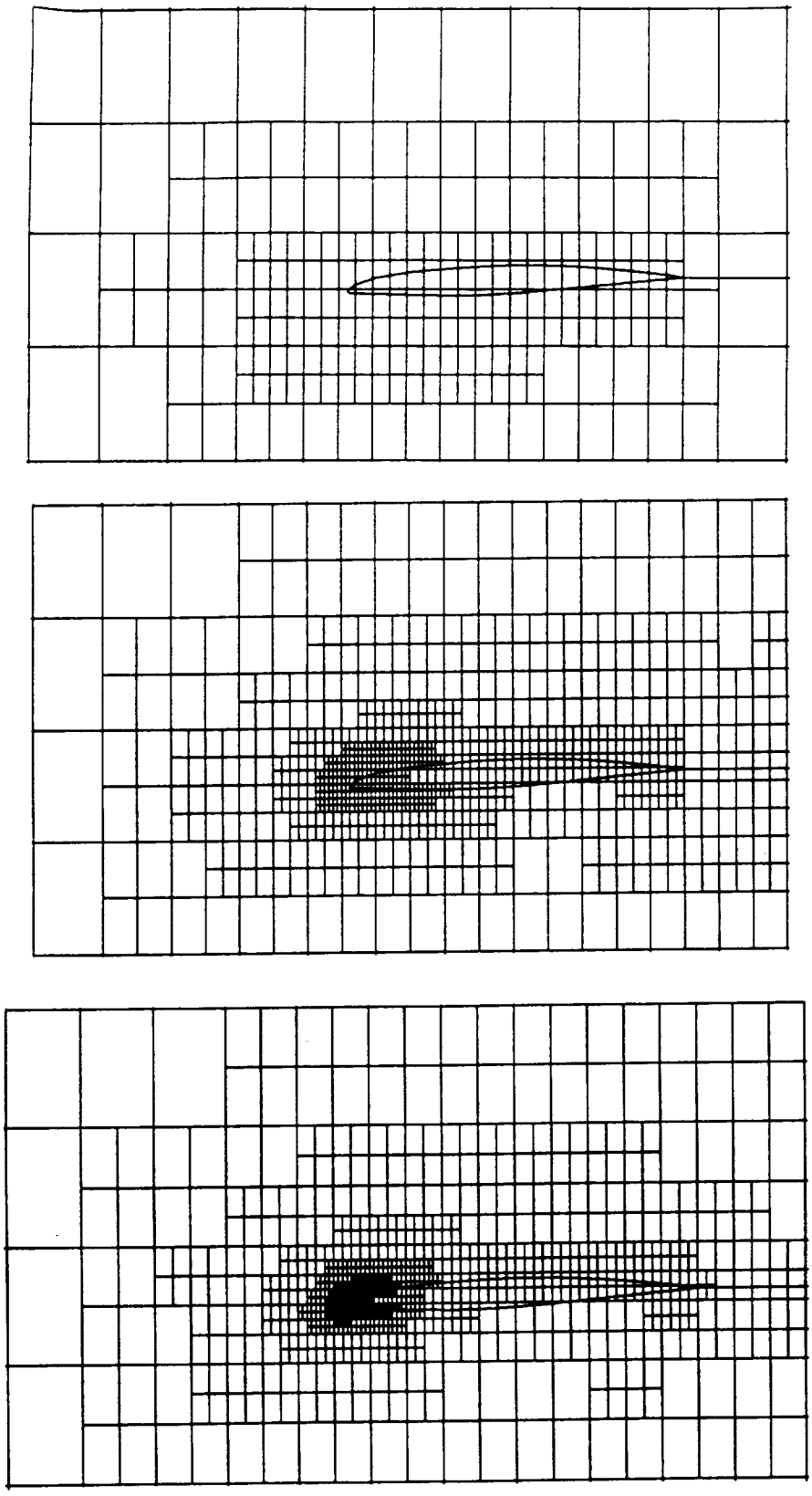


Figure 2.19: Initial Grid and two Grids Created in an Application of the Adaptive Method.

2.5 POSTPROCESSING

A finite element postprocessing capability has been implemented to smooth out irregularities in surface pressure distributions [69, 70]. The irregularities in the pressure distribution are illustrated in Figure 2.20 and arise due to the fact that the trilinear functions used to approximate the potential lead to essentially constant velocity in a given box. If more that one panel corner point is located in a given grid box then the pressure at these points (calculated from the velocity using the isentropic formula) also appears to be essentially constant.

To eliminate this apparent anomaly, a velocity $\overline{V}$ is computed for each unknown Φ or Ψ, located at a grid point using the following procedure.

- All regions influenced by the unknown are found.
- At the spatial location of the unknown the velocity basis functions of these regions are evaluated.
- $\overline{V}$ is given by the average of these velocity vectors.

To evaluate the velocity at any point in space, the region containing the point is found and the velocity components at the eight corner unknowns of the region are trilinearly interpolated.

For the sphere, the surface pressure is shown with and without the postprocessing step outlined above in Figure 2.20 for a linear flow case and a transonic case . The effect of post processing is evident.

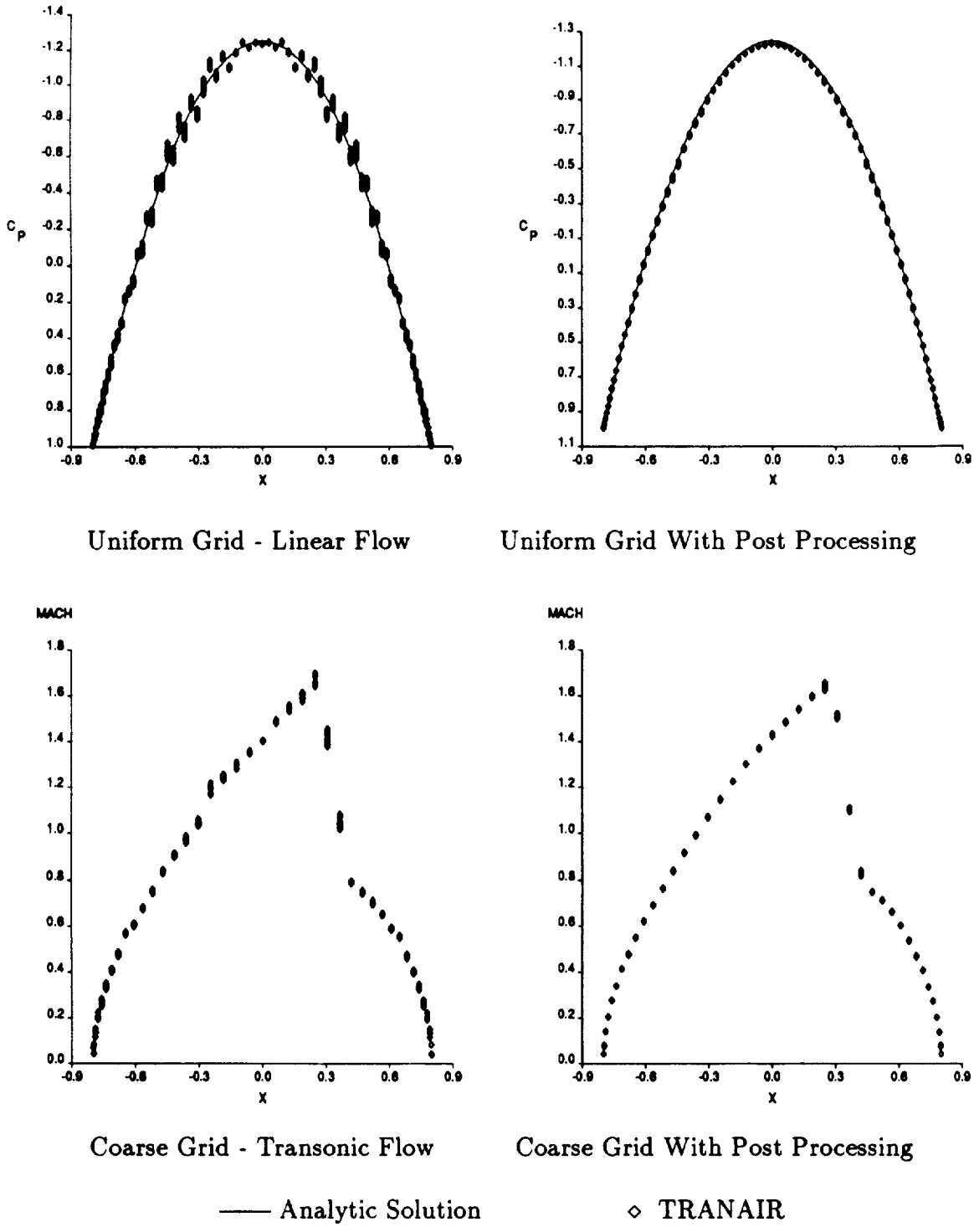


Figure 2.20: Solutions With and Without Post Processing for a Sphere in Linear Flow, $M_\infty = 0.0$, and Transonic Flow, $M_\infty = 0.7$.

2.6 SUPERSONIC FREE STREAM FORMULATION

The grid generation, discretization, and dissipation used to solve problems in supersonic free stream are essentially identical to that used in the problems in subsonic free stream. However, due to the difference in the character of the full potential equation in supersonic flow there are certain differences in the way the far field boundary conditions are imposed and the discrete equations are solved.

2.6.1 Far Field Treatment

The same governing equation is applicable in supersonic free stream flow. However the far field behavior of the flow is different from that in subsonic free stream flow. The source parameterization (Section 2.3.2) cannot be employed in the case of supersonic free stream flow. The Poisson solver constructed for the Prandtl Glauert equation (Appendix D) is no longer valid. Instead other types of boundary conditions are applied at the outer boundary of the computational domain.

Due to the hyperbolic nature of the flow initial conditions are required at the inflow boundaries (typically the upstream boundary). Since in supersonic flow the configuration has no upstream influence, the appropriate boundary condition is that the perturbation potential there be zero. In implementing this boundary conditions the perturbation is forced to be zero at two upstream planes of the global grid.

At the outflow boundaries of the computational domain no boundary conditions are required in principle, because the solution can be obtained in a marching process. However, since the present method uses a sparse solver on a reduced set that includes all the unknowns in the field it is essential to impose some boundary conditions that do not feed their influence upstream. In this case a $\phi_{xx} = 0$ boundary condition is imposed at the downstream boundary.

On the side boundaries the initial conditions at the upstream boundary and the ϕ_{xx} boundary conditions imply that the perturbation potential at the side boundaries also be zero.

It is noted that the imposition of these conditions on the outer boundary of the grid may cause shock reflections which could influence downstream portions of the flow field, particularly at low supersonic free stream Mach numbers when the Mach cone angles are large. A better approach would be to assume supersonic linear flow outside the computational grid and impose a conical flow condition at the outer boundary. This has not yet been implemented.

2.6.2 Solution of Discrete Equations

As mentioned earlier, all equations are included in the sparse matrix preconditioner, including the equations corresponding to the global grid points away from regions of local grid refinement and away from the configuration surface. To take advantage of the hyperbolic nature of the flow, the equations in the sparse solver are ordered by

increasing x coordinate value, rather than by nested dissection. The system is solved by the nonlinear GMRES procedure exactly as is done in the subsonic free stream case.

For the supersonic free stream case, the convergence of the linear problem in the Newton method is essentially the same as in the subsonic free stream cases. The non-linear Newton iterations converge better with grid sequencing and viscosity damping. However, on occasion supersonic free stream cases have been observed to "stagnate" for a few cycles before converging further. In addition, when using solution adaptive gridding, it has been observed that more robust convergence behavior occurs if viscosity damping is used on each successive grid. This requires more iterations on the finer grids before turning off the extra viscosity, but provides more reliable convergence.

The solution adaptive grid features of TRANAIR are the same in supersonic free stream flow as in subsonic free stream flows. Normal shocks and expansion regions are easily detected and grid is generated to resolve gradients. When the shocks are oblique, it takes more cycles of solution adaptivity to begin to detect their presence. Figures 2.21 through 2.24 illustrate a sequence of grids generated about a sphere-cone configuration.

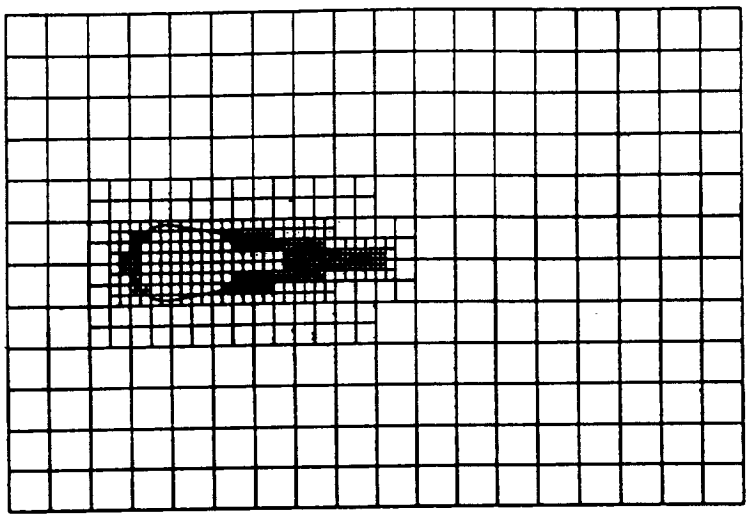


Figure 2.21: Solution Adaptive Grid (No. 1) for the Supersonic Cone

The early solution adaptive grid refinements concentrate on the gradients in the expansion region near the upstream stagnation point on the spherical face. Three cycles of solution adaptation were required to get errors in the stagnation region reduced sufficiently so that the bow shock was well-recognized. After five cycles of grid refinement the bow shock is clearly developed up to the point where it becomes quite oblique. At that point, the grid is too coarse to sufficiently resolve the oblique shock and it diffuses badly. The relative distribution of the computed local error estimates indicates that the next cycles of solution adaptive refinement will better resolve the oblique portions of the bow shock, but it is clear that it is desirable to

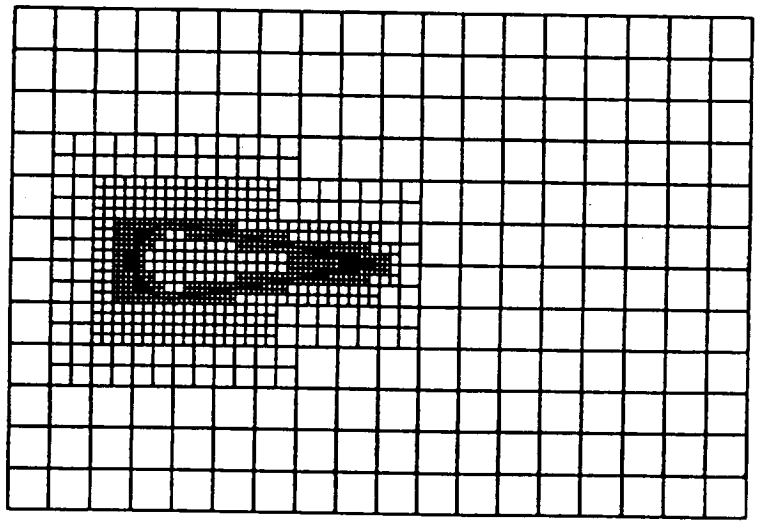


Figure 2.22: Solution Adaptive Grid (No. 2) for the Supersonic Cone

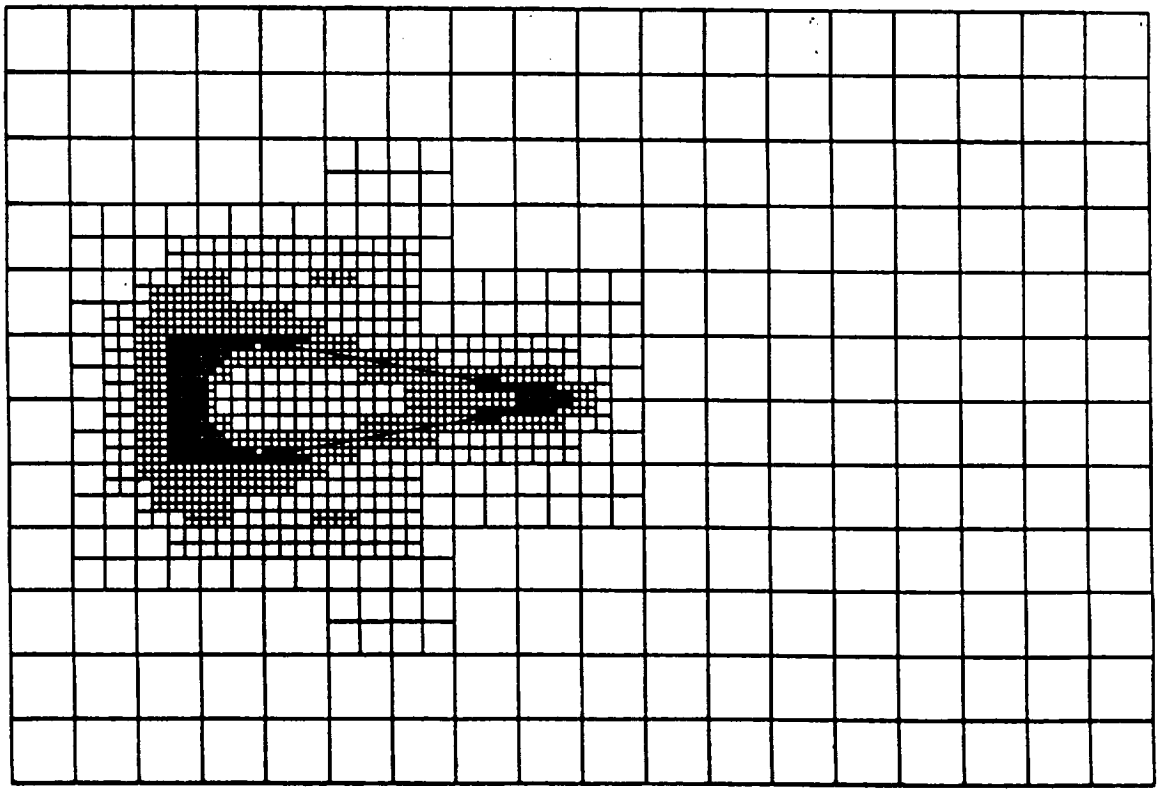


Figure 2.23: Solution Adaptive Grid (No. 3) for the Supersonic Cone

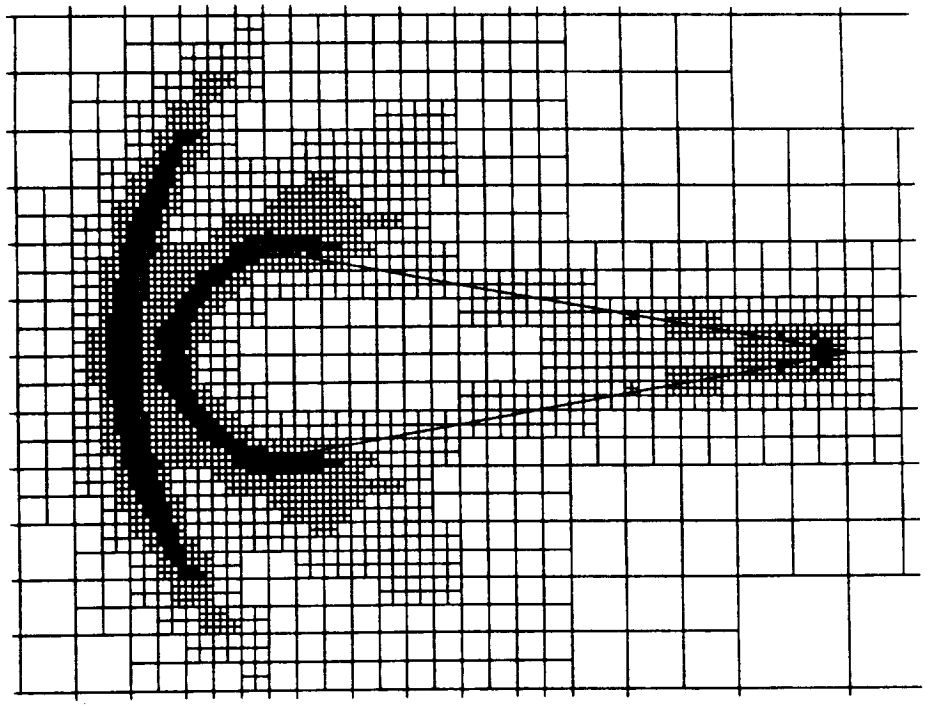


Figure 2.24: Solution Adaptive Grid (No. 5) for the Supersonic Cone

modify the solution adaptive grid strategy in a way to allow it to detect oblique shocks earlier in the solution adaptive gridding cycles.

2.7 PROGRAMMING CONSIDERATIONS

The numerical method incorporated in TRANAIR has many algorithms. In implementing these algorithms a significant amount of effort was spent ensuring that these algorithms make efficient use of vector supercomputer hardware features. Since no assumptions were made regarding availability of machines with extremely large central memory, the code was designed to use out-of-core storage, e.g., the Cray SSD. (TRANAIR also can be easily modified for in-core applications.) This also required that special attention be paid to memory management and input/output issues. Data structures are of paramount importance to any application code and TRANAIR is no exception. Due to the unstructured nature of the grid and large amount of data to be handled, TRANAIR uses some unique data structures. Finally, it is also worth noting that the code has been developed by a team of people. To facilitate team work and minimize maintenance problems, "black box" (modular) coding was used where possible, leading to a large collection of library routines performing various standard functions.

In the following these issues are discussed in some detail. No attempt is made to provide a complete list of subroutines nor to discuss any specific algorithm at great length. The purpose of this section is to set forth ideas that have gone into building the modules that make up the TRANAIR code.

2.7.1 Memory Management

To maximize the size of the problem that can be solved with a given amount of memory it is imperative that the available central memory be used efficiently. This issue becomes especially important when many programmers are involved in coding different modules and need to access the same memory locations.

To resolve this issue, a self contained memory management system was developed. Most of the memory space used in the code is contained in a single large one dimensional scratch array. This array is divided into smaller arrays as needed in any subroutine. The remaining portion of the big array is passed down through the calling sequence of any subroutine needing further scratch space. The latter routine can then further subdivide the array as needed. The subdivision is hierarchical and is implemented using several FORTRAN subroutines. When storage for an array is requested, an identifier and the length of the array are supplied. A pointer to the array is returned. This allows reference to the array even after "garbage collection" (defined below). Array storage is freed when no longer needed. When storage for an array is requested, if possible, a consecutive block of storage is found and allocated. However, if storage for many arrays have been allocated and de-allocated the available storage may be fragmented with no sufficiently long consecutive block of storage in the big scratch array. In this case, storage in the scratch array is garbage collected, i.e., all allocated array storage is moved to the front of the scratch array (the pointers are changed) leaving a contiguous block of available storage at the back. If this block is still not large enough, the program will abort. The simple remedy then is to increase the dimension of the main scratch array.

2.7.2 Input/Output

TRANAIR uses a centralized I/O system for temporary files. The temporary files are typically used to store data so that central memory can be freed for some other purpose. The data stored on such files includes that for the oct-tree data structure, the boundary operators, the GMRES search directions, and the decomposition of the Jacobian matrix used as a preconditioner. These files are made to reside on the SSD, the disk, or in central memory if available. For most purposes these files are treated as temporary files that are generally lost at the end of program execution.

Data I/O is carried out through special routines. Some of the bigger datasets are written using unblocked FORTRAN I/O. The datasets are accessed using unit numbers which are determined and stored within the program. When the dataset under question has outlived its utility the unit can be closed, thus freeing it for other use as needed. This process is automatic.

A good example of I/O usage is the residual computation procedure. The residual calculation requires the potential at unknown locations (nodes) and density at the centroids of regions. In addition, quantities such as the velocity, switching function, etc. are also required at region centroids. If all these quantities could be held in core at one time the computation of residuals would indeed be extremely fast. However, since the available central memory on many Cray computers is small, some of the data is blocked and stored on a mass storage device and computations are performed one block at a time. The field quantities (those defined at the nodes) are held in core as fields and the region based quantities are blocked and brought into central memory as the computations proceed. The I/O required in these operations is performed using the central I/O system.

In the case of subsonic flow, two passes through the "blocks" are necessary to compute residuals. First, with the array of potential values at unknown locations stored in core, velocity operators and corner unknown indices are brought into core a block at a time. For each block, velocity components at element centroids and hence densities are computed and stored, before considering the next block. This step easily can be vectorized with vector length equal to the length of the block. Second, with arrays of potential and residual values in core, divergence operators, densities at centroids and corner unknown indices are brought into core a block at a time and used to scatter contributions to residuals at the corners.

Other examples of blocking are found in generating and decomposing the sparse solver matrix (see Appendix E).

2.7.3 Vectorization Issues

The locally refined structure of the computational grid and the need to achieve good performance on vector machines makes many aspects of the coding more complex. The fundamental change from the basic structure of a uniform Cartesian grid code [39] and many logically rectangular body fitted grid codes is that many of the operations involve *indexed* (indirectly addressed) arrays, i.e., in many instances, instead of *directly* addressed vector operations such as

```
X(I) = X(I) + C * Y(I),
```

vector operations are required of the form

```
X(I) = X(I) + C * Y(IND(I))
```

or

```
X(IND(I)) = X(IND(I)) + C * Y(I).
```

With recent Cray compilers it is possible to vectorize loops with such operations when no vector dependencies exist. Through use of careful coding to avoid the need for many double-indexed arrays, e.g., arrays of the form Y(IND1(IND2(I))), such operations in TRANAIR have been made to execute at high rates (typically at a third to a half of the peak rate possible with direct addressed arrays). Much use of this is made in the sparse matrix decomposition and forward and backward substitution phases (see Appendix E).

The residual computation also uses indexed operations. As described in Section 2.4 and Appendix B, finite element operators and velocity operators are computed for each element. These operators depend only on the geometry of the element. When the element is in a box not cut by a boundary, only the grid refinement level (relative to the global grid) of the box is needed to define the operator. D-region operators are computed and stored out-of-core. In order to apply such an operator it is necessary to know the indices of unknowns located at all eight corners of the element. Because of local grid refinement, these corner unknowns are *not* stored in a contiguous manner, thus making the resulting operations indexed.

The residual calculations consist of two phases: gathering information for each region and scattering information from the regions to the corner nodes. In the first phase, velocity, density and switching function values are computed at the centroids of every region. At the ith region centroid, the density ρ_i is obtained from the magnitude of velocity, the kth component of which is computed via

$$v_i^k = \sum_{j=1}^{8} V_{ij}^k \Phi(IRU(i,j)), \tag{2.61}$$

where Φ is potential, V_{ij}^k is the operator coefficient giving the contribution to the kth velocity component at the centroid of the ith region from the jth corner unknown of the region, and $IRU(i,j)$ is the index of the jth corner unknown of the ith region. The coefficients V_{ij}^k for D-regions are stored out-of-core. While Φ is held in core, velocity components are computed for a block of regions at a time. By ordering elements not cut by boundaries (standard regions) in separate blocks, each block consisting of regions at the same level of grid refinement, the FORTRAN loop implementing Eq. (2.61) involves single-indexed arrays and executes at a rate approaching 100 megaflops on a single-processor Cray X-MP (having 9.5 nanosecond cycle time).

When some or all of the regions have supersonic flow, an extra step in the first phase of the residual calculation is necessary for each block to incorporate upwinding effects. The upwinded density, $\tilde{\rho}$, in region number it is given by

$$\tilde{\rho}_{it} = \rho_{it} + \mu_{it} \sum_{if=1}^{6} SF_{if}(v_{it}) \sum_{ir=1}^{4} C_{it,if,ir}(\rho_{IBB(it,if,ir)} - \rho_{it}), \tag{2.62}$$

where $\{C_{it,if,ir}\}$ are the operator coefficients and $\{IBB(it,if,ir)\}$ are the indices of regions adjacent (*viz.* index ir) to region number it across element face number if. The symbol μ in Eq. (2.62) denotes the switching function and SF is a blending function (see Eq. (2.43)). In these equations the doublet parameters have been omitted for simplicity. The calculation of region centroid values of $\tilde{\rho}$ is done by blocks of regions, as was described above for values of ρ, with Φ stored in core. The FORTRAN loop implementing Eq. (2.62) is over all supersonic regions in a block (i.e., over block regions on which μ is greater than zero). In the loop, the region number it of Eq. (2.62) is obtained via $it = IND(I)$, where I is the loop counter ranging from one to the number of supersonic regions in the block. This means the array $IBB = IBB(IND(I),\cdot,\cdot)$ is an indexed array, and, consequently, the FORTRAN array representing $\rho_{IBB(it,if,ir)}$ of Eq. (2.62) is double-indexed. The peak execution rate for this FORTRAN loop on a single-processor Cray X-MP (having 9.5 nanosecond cycle time) is about 50 megaflops. However, the work done in this loop accounts for an insignificant portion of the total work done in an application.

In the second phase of the residual calculation, the region-centered quantities obtained in the first phase are used to compute the residuals associated with the nodal solution unknowns. This phase is the dominant cost in the residual calculation and so efficiency is very important. The residual R_l for the lth solution unknown is obtained via

$$R_l = \sum_{region\ i} \rho_i \sum_{j=1}^{8} D_{ijl}\Phi(IRU(i,j)) + \sum_{region\ i} (\tilde{\rho}_i - \rho_i) \sum_{j=1}^{8} D_{ijl}\Phi(IRU(i,j)) \tag{2.63}$$

where D_{ijl} is the finite element divergence operator coefficient contribution to the lth unknown for the ith region from the unknown at corner j. The FORTRAN implementation of Eq. (2.63) uses two nested loops, reversing the order of the summations so that the outer loop is over the eight corner unknowns and the inner (vectorizable) loop is over the elements of the block. Vectorization is possible because for any outer loop index j, the jth corners of all the regions in the block are distinct, and so no vector dependency occurs. This would not be true for an arbitrary block of D-regions, but in TRANAIR, the corners are made distinct within a block by separating possible duplicates into different blocks.

Careful design of these algorithms was necessary to minimize storage, allow effective use of the Cray SSD, and achieve reasonable CPU speed.

2.7.4 Data Structures

TRANAIR uses a number of different data structures to facilitate compact and efficient usage of data. A prime example is the oct-tree data structure discussed at length in Appendix A. Among the other data structures used are the region-unknown lists, box-neighbor lists, operators, etc.

2.7.5 Program Libraries

Wherever possible "black box" (modular) coding has been used to increase flexibility. An example of "black box" coding is the nonlinear GMRES routine which sees the entire residual evaluation process as an arbitrary function to be calculated. The code is built up from a set of libraries containing groups of subroutines which can be classified together. These consist of libraries for:

- input processor
- solver
- output processor
- special purpose mathematical routines
- sparse solver
- Green's function
- general purpose utility routines
- general purpose mathematical routines
- abutment processor
- fluid dynamics routines

Chapter 3

RESULTS

Many results of applying TRANAIR are presented in this chapter. These results demonstrate the ability of the code to handle general geometry configurations in subsonic and supersonic free stream, the reliability with which these solutions can be obtained, and the flexibility of the code in allowing modeling of various flow features such as leading edge expansions, weak normal shocks, oblique shocks, and regions with different total temperatures and pressures. The example results are divided into three groups according to flow type.

The first consists of cases in subsonic free stream governed by the linear Prandtl-Glauert equation. These cases are presented primarily for the purpose of comparing results with those of panel methods and analytic solutions where available. Neither Newton method damping nor grid sequencing is required in these cases. In each TRANAIR run, a single grid was constructed based on user specifications.

The second group consists of cases in subsonic free stream where it is necessary to solve the full potential equation because the flow characteristics are nonlinear and possibly transonic. Results are presented for TRANAIR runs that employed single grids, grid sequencing, and solution adaptive grids.

The third group consists of cases in supersonic free stream where again the solution of the full potential equation is necessary. The flow is predominantly hyperbolic in character. Subsonic regions and transonic flow characteristics are present in some cases. Solution adaptive grids were employed in all the runs in this group.

The results presented in this chapter have been obtained over a period of two years. Where possible the results obtained from the most recent versions of the code are presented. In every case the result can be repeated to the same or improved accuracy with the most recent version of the code.

In all cases presented in this paper, the solution (primarily represented by static pressure) is displayed at panel corner points. The static pressure is generally represented by its non-dimensional counterpart defined as

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty q_\infty^2} \tag{3.1}$$

where p is local pressure, ρ_∞, p_∞, and q_∞ are the free stream density, local pressure, and velocity magnitude.

The results presented here are obtained on the Cray X-MP machine with up to 4MW of central memory and up to 128 MW of SSD storage or the Cray Y-MP.

3.1 RESULTS FOR LINEAR FLOW

In this section, linear flow solutions are discussed. Results for a sphere, the ONERA M6 wing, and the F16 fighter aircraft configuration are presented.

3.1.1 Sphere

For the sphere in incompressible flow, an analytic solution is available. This is a nontrivial problem for a Cartesian grid method since the surface intersects the grid in many different ways. A sphere with radius 0.8 was analyzed at $M_\infty = 0$ and $\alpha = 0.0°$. Four grids were used to test the accuracy of TRANAIR. In Figure 3.1 the paneling used to describe the sphere surface in the coarse and medium grid cases is shown (there are 1600 panels describing the geometry of the half sphere using one plane of symmetry). With the fine and uniform grids, the paneling was doubled in each direction. Planar cuts through the four grids are shown in Figure 3.2. The uniform grid had 123,680 elements. It is noted that the outer boundary of the computational domain in the uniform grid case is very close to the boundary. The coarse, medium, and fine grids shown have 10356, 35456, and 149,515 elements respectively. Stagnation regions (those totally inside the sphere) are not included in the element totals. These cases were run with one plane of symmetry. Only half of each cut is shown since each cut is symmetric about a second plane of symmetry.

In Figure 3.3 the surface pressure[1] for the sphere is plotted as a function of x. Also shown is the corresponding analytic solution. Data at all circumferential stations are plotted. For the 1600 panel case, there are 20 stations at each x value. The scatter of surface pressure at a constant x coordinate is due to the use of Cartesian grid and provides a good measure of the overall accuracy. The solution accuracy improves significantly as the grid is refined. The expected quadratic convergence rate in potential as the grid is refined has been verified earlier [36] in this case.

[1]In all the subsequent discussion the term *surface pressure* is used to indicate the pressure coefficient

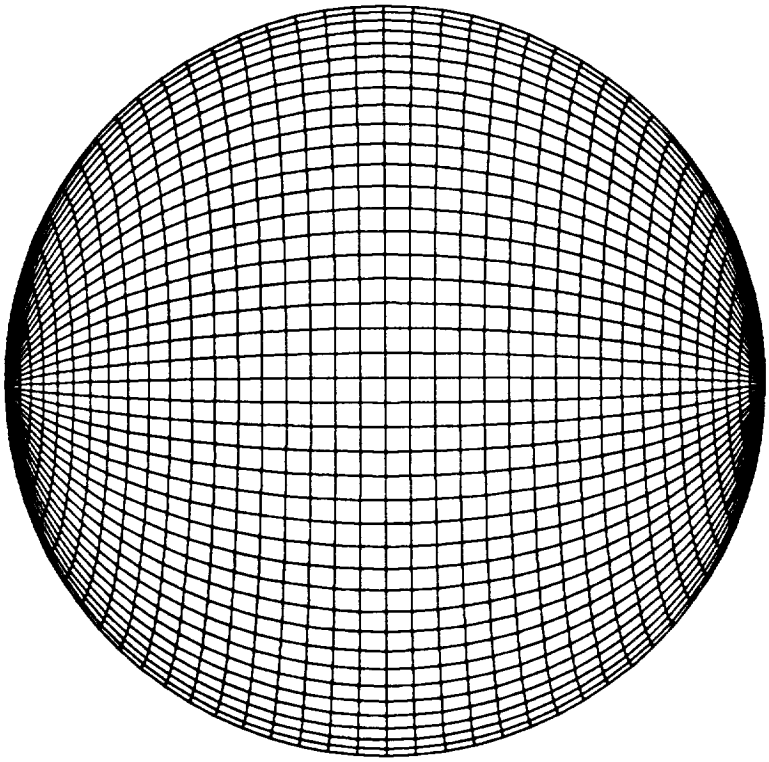


Figure 3.1: Paneling Used for Sphere in Linear Flow, 1600 Panels.

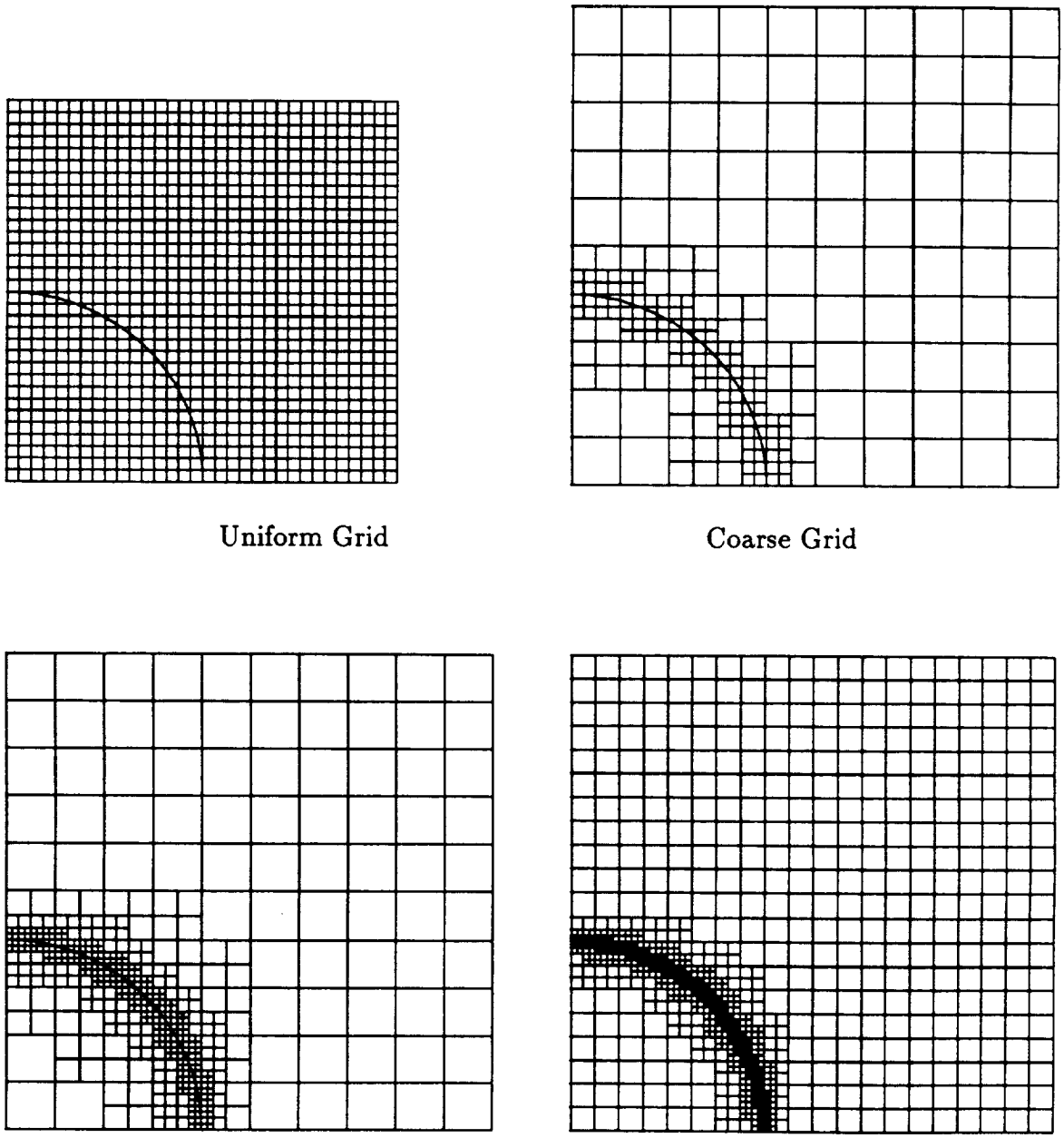


Figure 3.2: Cuts Through Four Grids for a Sphere in Linear Flow, $M_\infty = 0$.

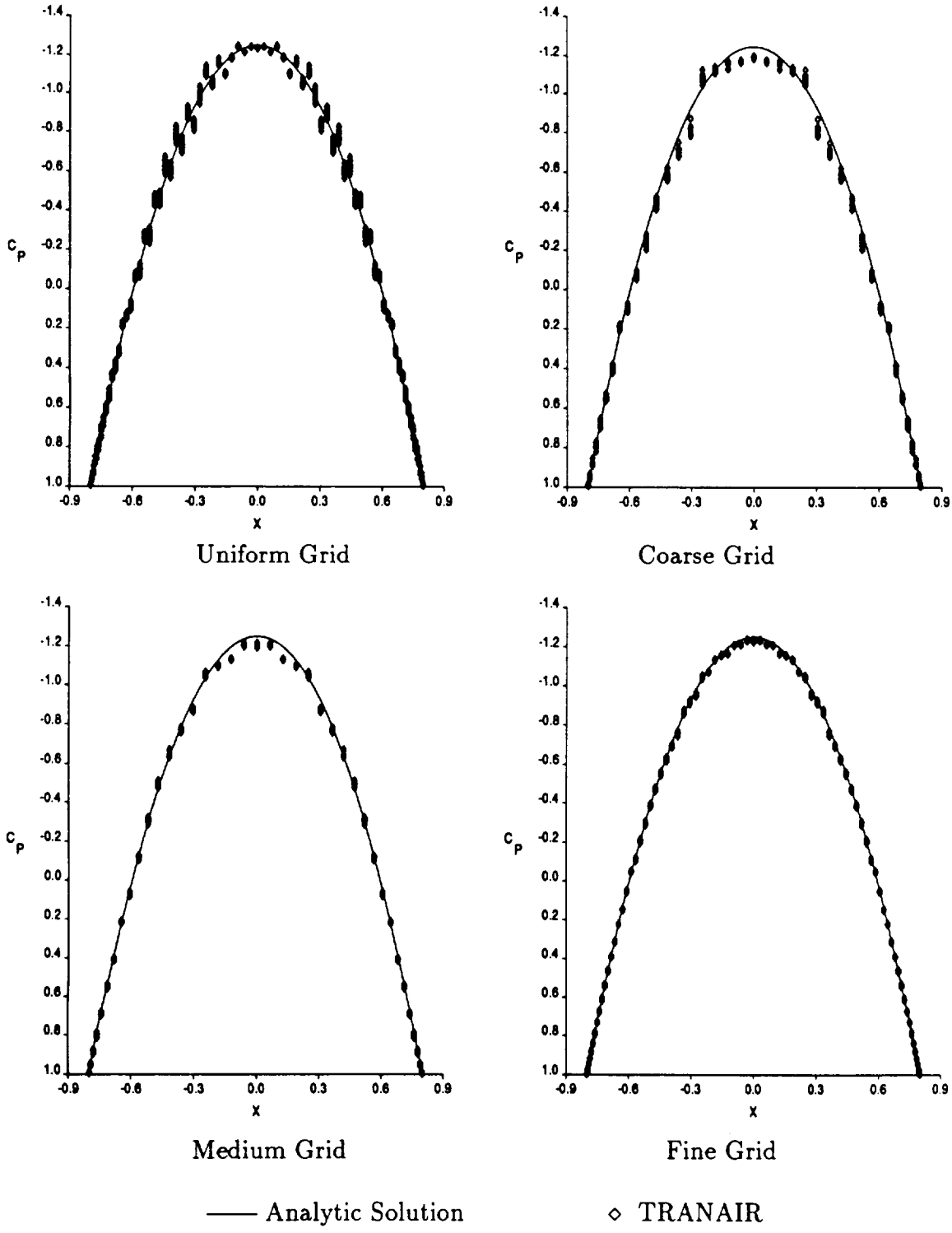


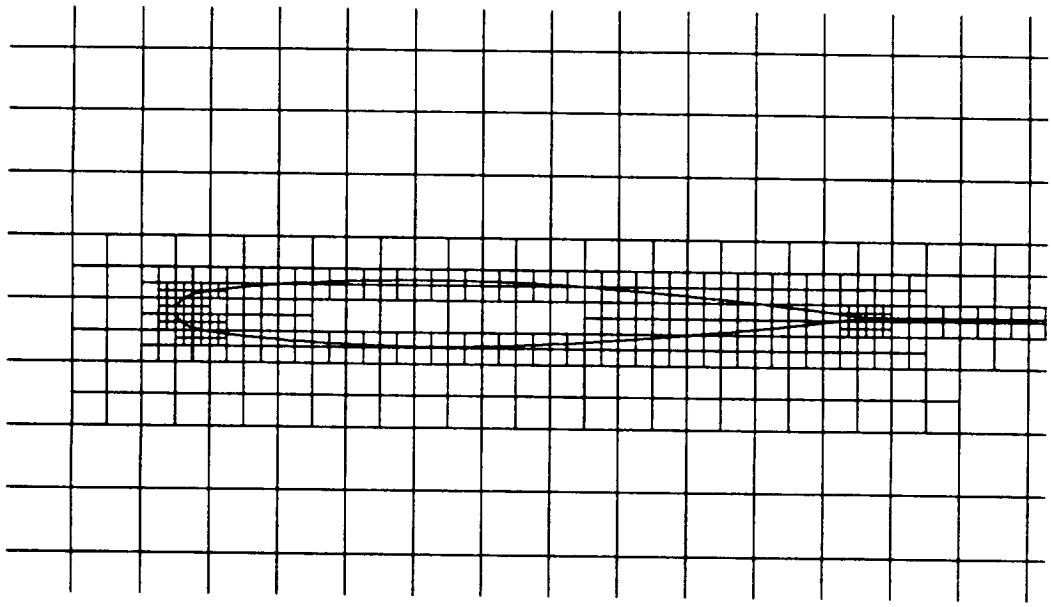
Figure 3.3: Solutions on Four Grids for a Sphere in Linear Flow, $M_\infty = 0$.

3.1.2 ONERA M6 Wing

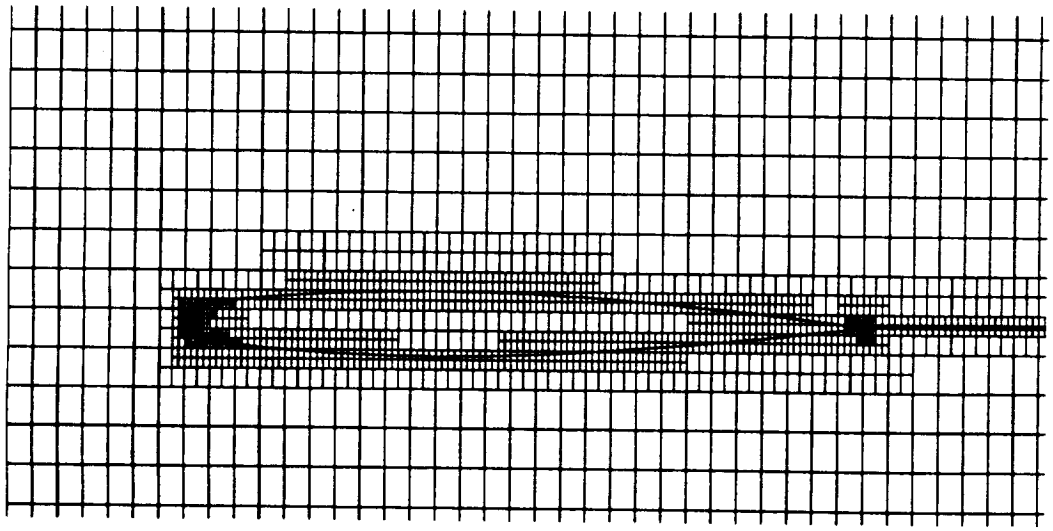
An ONERA M6 wing is analyzed at $M_\infty = 0$ and angle of attack $\alpha = 3.06°$. The boundary is described by 1800 panels (see Figure 3.6). The panels have a very high aspect ratio, being much longer in the spanwise direction than in the chordwise direction. This paneling is adequate for a solution in linear flow because the solution also changes more rapidly in the chordwise direction than in the spanwise direction.

TRANAIR was run on a coarse grid having 35,188 elements and a fine grid having 249,305 elements. Vertical cuts through the two grids at the plane of symmetry are shown in Figure 3.4. Figure 3.5 shows a waterline cut through the coarser grid. The clustering of fine grid cells at the leading and trailing edges is necessary to resolve high velocity gradients.

Figure 3.6 compares surface pressure at the 20% span station with a solution obtained with a panel method. Note that the fine grid TRANAIR solution agrees well with the panel method solution. The leading edge is enlarged in the third plot. Figure 3.7 shows two other stations from these same solutions.



Coarse Grid



Fine Grid

Figure 3.4: Cuts Through Two Grids for the ONERA M6 Wing in Linear Flow, $M_\infty = 0$, $\alpha = 3.06°$.

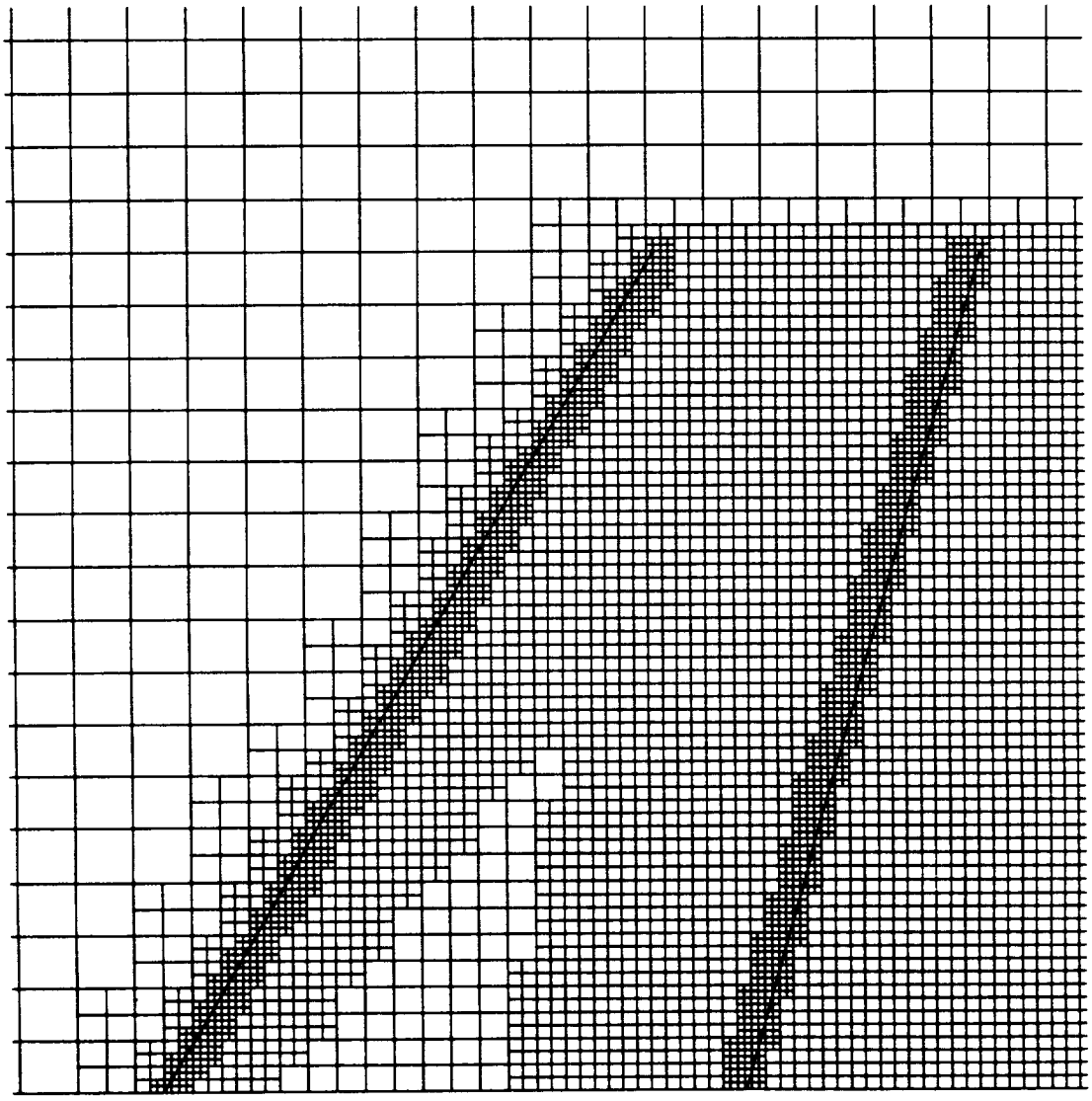


Figure 3.5: Waterline Cut Through ONERA M6 Coarse Grid.

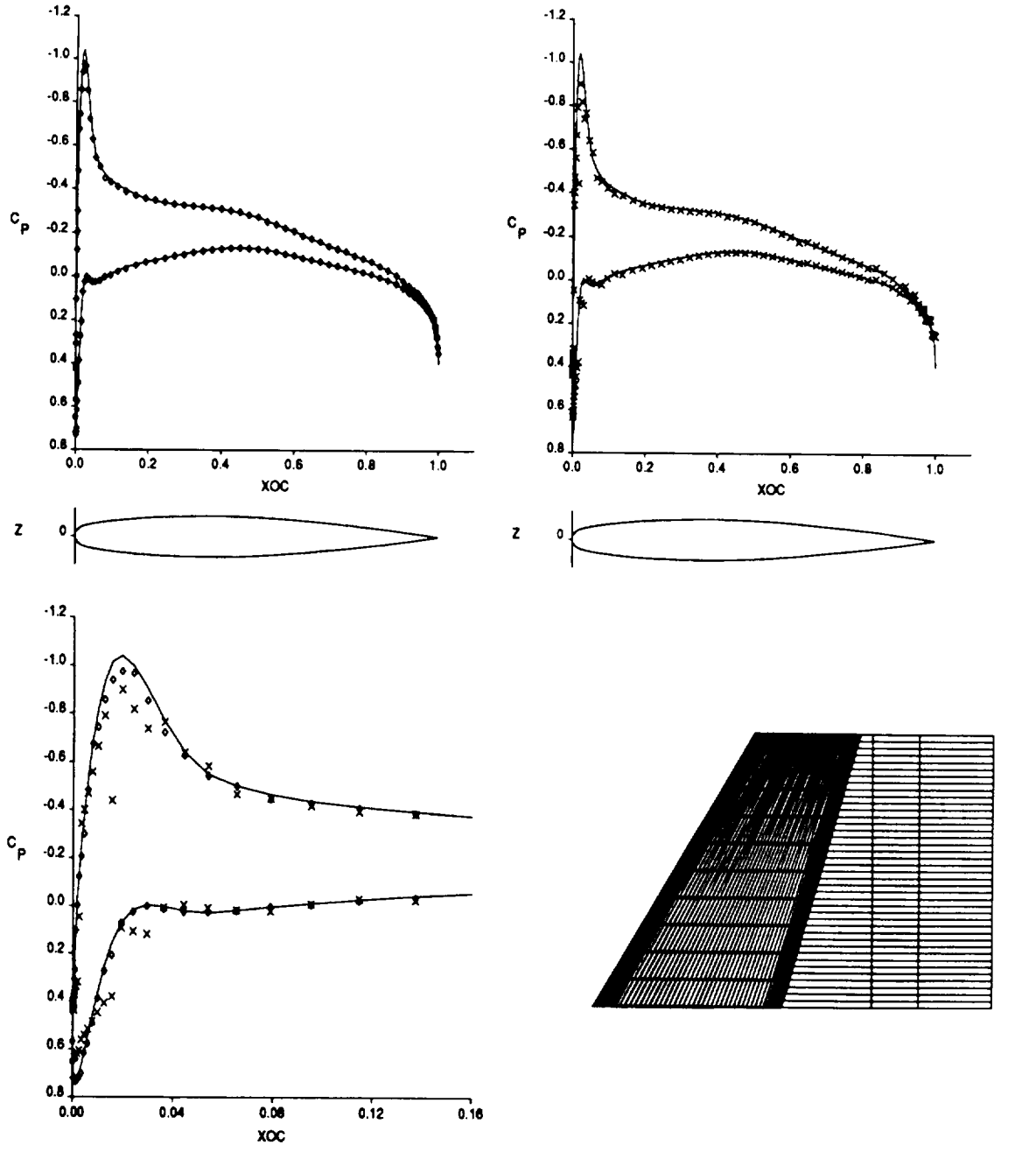
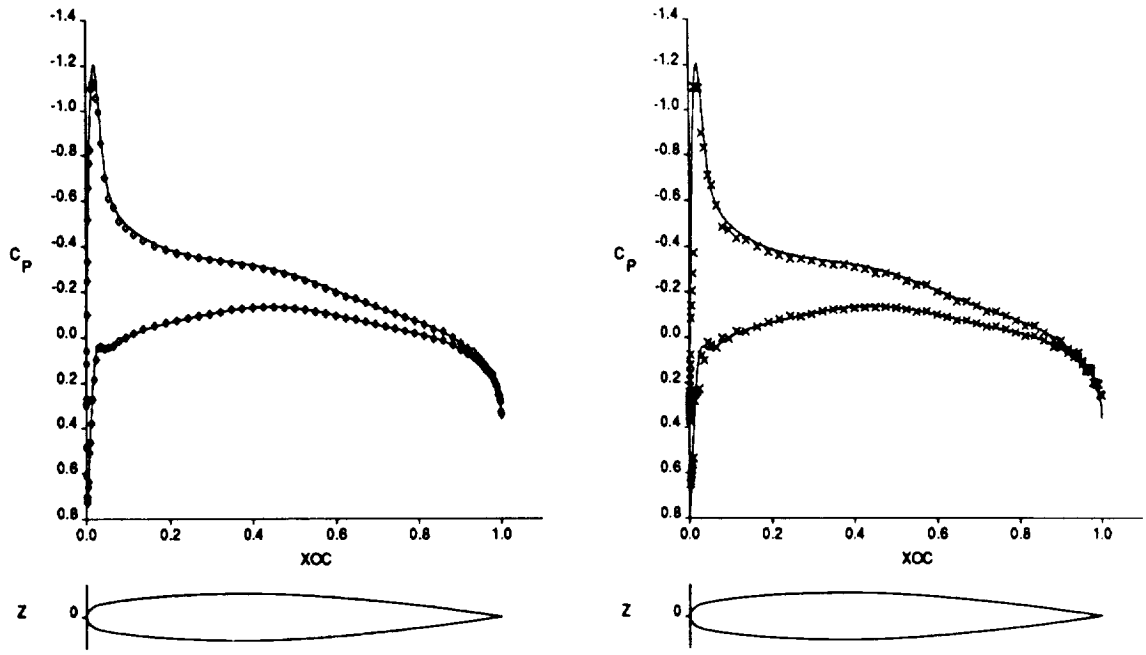
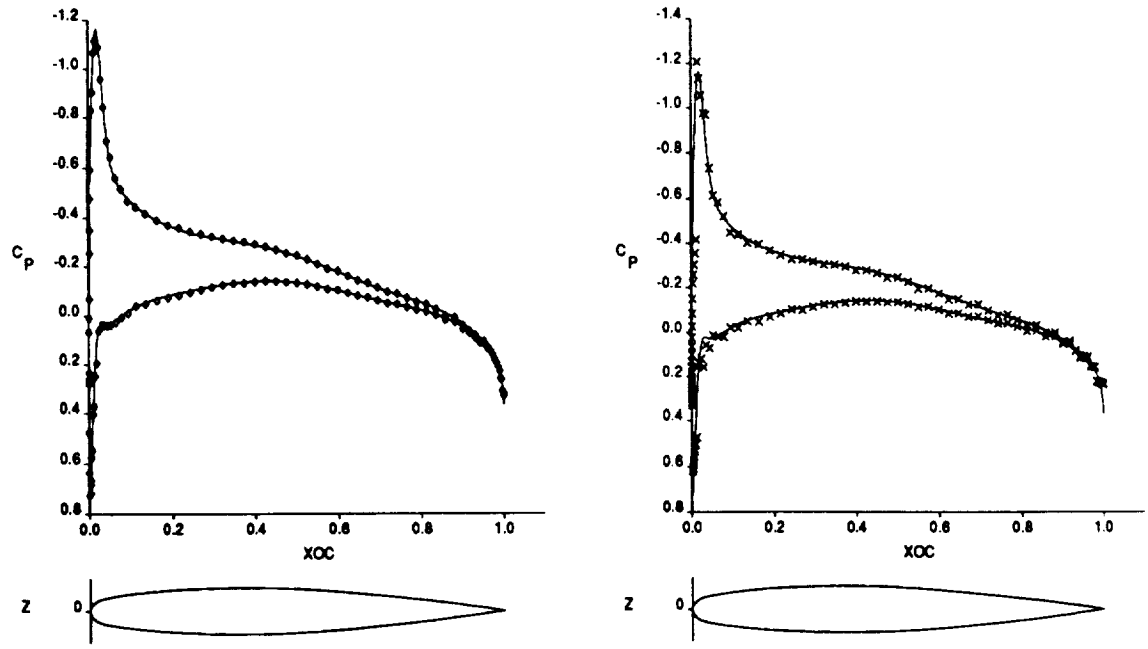


Figure 3.6: Three Solutions for the ONERA M6 Wing in Linear Flow at 20% span, $M_\infty = 0$, $\alpha = 3.06°$.



60% Span Station



80% Span Station

—— Panel Method ◇ TRANAIR Fine Grid × TRANAIR Coarse Grid

Figure 3.7: Three Solutions for the ONERA M6 Wing in Linear Flow at 60% and 80% Span, $M_\infty = 0$, $\alpha = 3.06°$.

3.1.3 F16 Fighter Aircraft

An F16 fighter aircraft configuration shown in Figure 3.8 was analyzed at $M_\infty = 0.6$ and $\alpha = 4.0°$. The configuration has 3510 panels. The TRANAIR run had 162,850 elements. Figure 3.9 compares surface pressure on the wing at two stations with those obtained using a panel method.

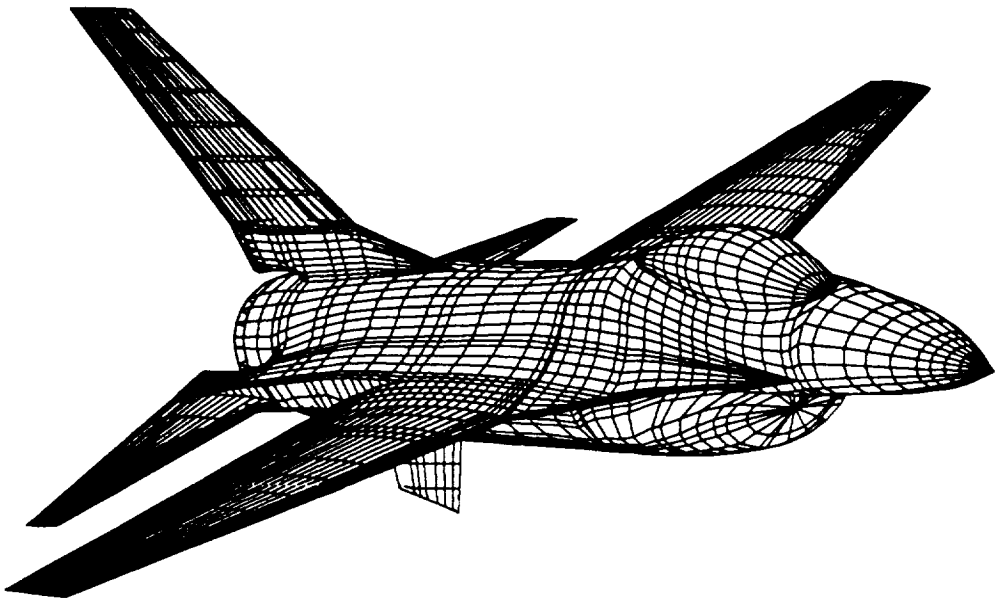


Figure 3.8: F16 Aircraft Configuration.

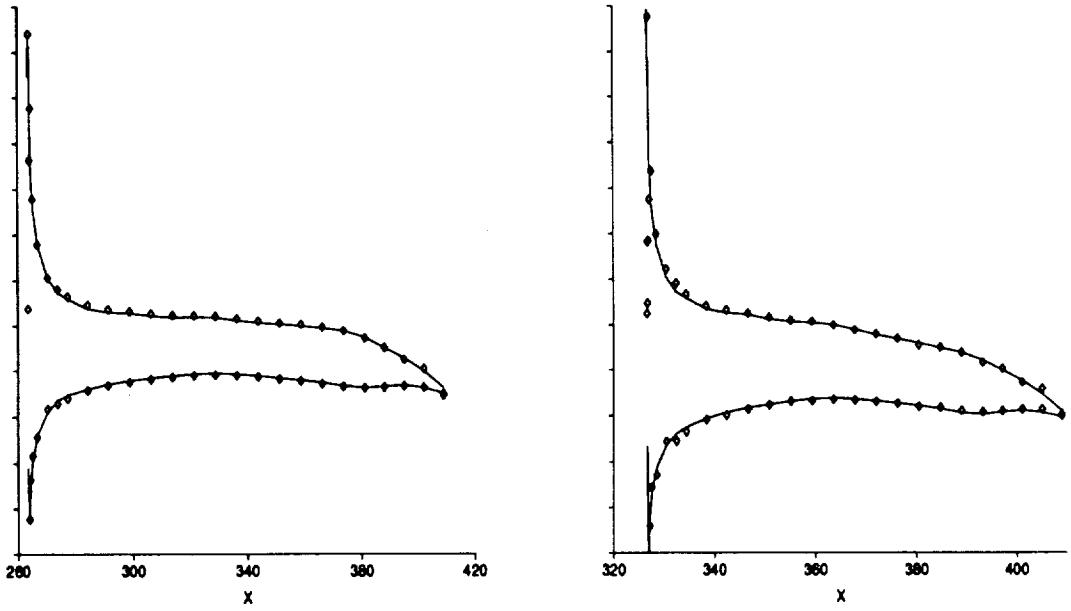


Figure 3.9: Surface Pressure at Two Stations on the F16 Wing, $M_\infty = 0.6$, $\alpha = 4.0°$.

3.2 RESULTS FOR NONLINEAR FLOW

To demonstrate the nonlinear capabilities of the method, solutions for a wide variety of three dimensional configurations are presented. These include complex fighter and transport configurations as well as the geometries used in Section 3.1 above for linear flow.

3.2.1 Sphere

A sphere with radius 0.8 is analyzed at $M_\infty = 0.7$. At this condition, the flow is transonic and contains a strong shock. This case was used to test the effectiveness of the upwinding used in TRANAIR. A fine grid was used to test the accuracy of the TRANAIR discretization. The grid contained about 170,000 elements and two planes of symmetry were used to reduce the size of the problem. A cut through this grid is shown in Figure 3.10.

Figure 3.11 shows the convergence history for this case with grid sequencing and with viscosity damping described in the previous section. Five continuation steps were needed to achieve convergence with viscosity damping in this case. Significant step size damping was required for the first viscous problem. No step size damping was needed with grid sequencing. There is no significant difference in the aerodynamic solution obtained via viscosity damping or grid sequencing.

Figure 3.12 shows surface Mach numbers as a function of x. Values at all circumferential stations are plotted. Because of the symmetry of the geometry and lack of angle of attack the solution should be axially symmetric. The TRANAIR solution is quite symmetric and also captures the well known re-expansion phenomenon at the foot of the shock. Post processing described in Section 2.5 was used.

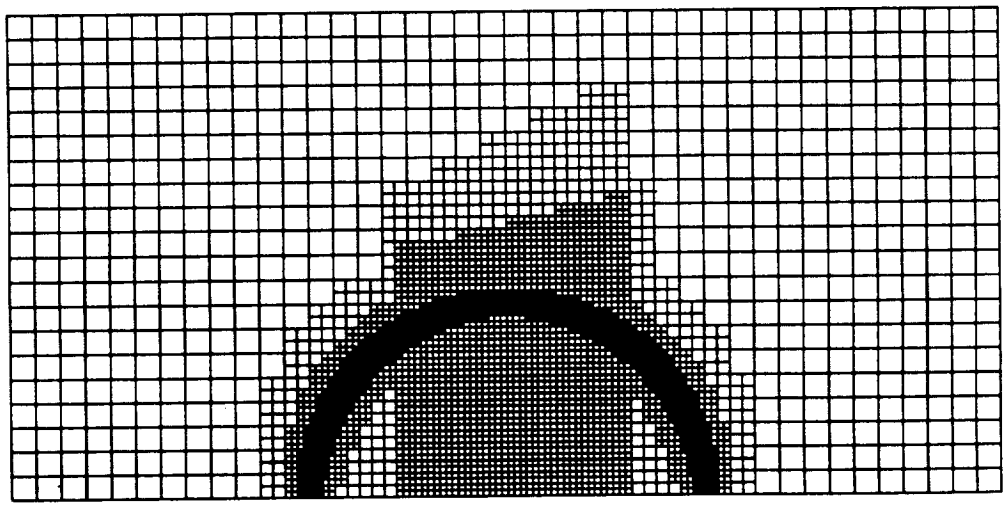


Figure 3.10: Cut Through the Grid for a Sphere in Transonic Flow, $M_\infty = 0.7$.

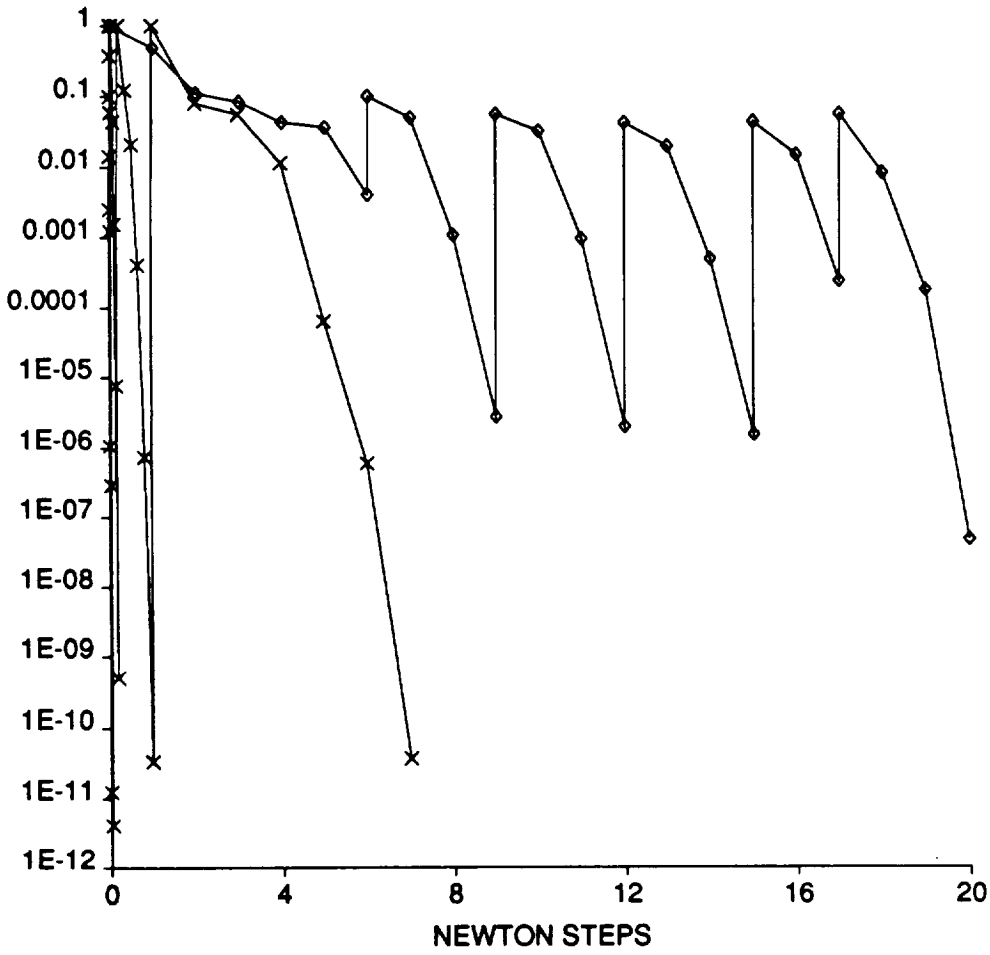


Figure 3.11: Convergence Histories for Viscosity Damping Method and Grid Sequencing Method for Sphere Case, $M_\infty = 0.7$.

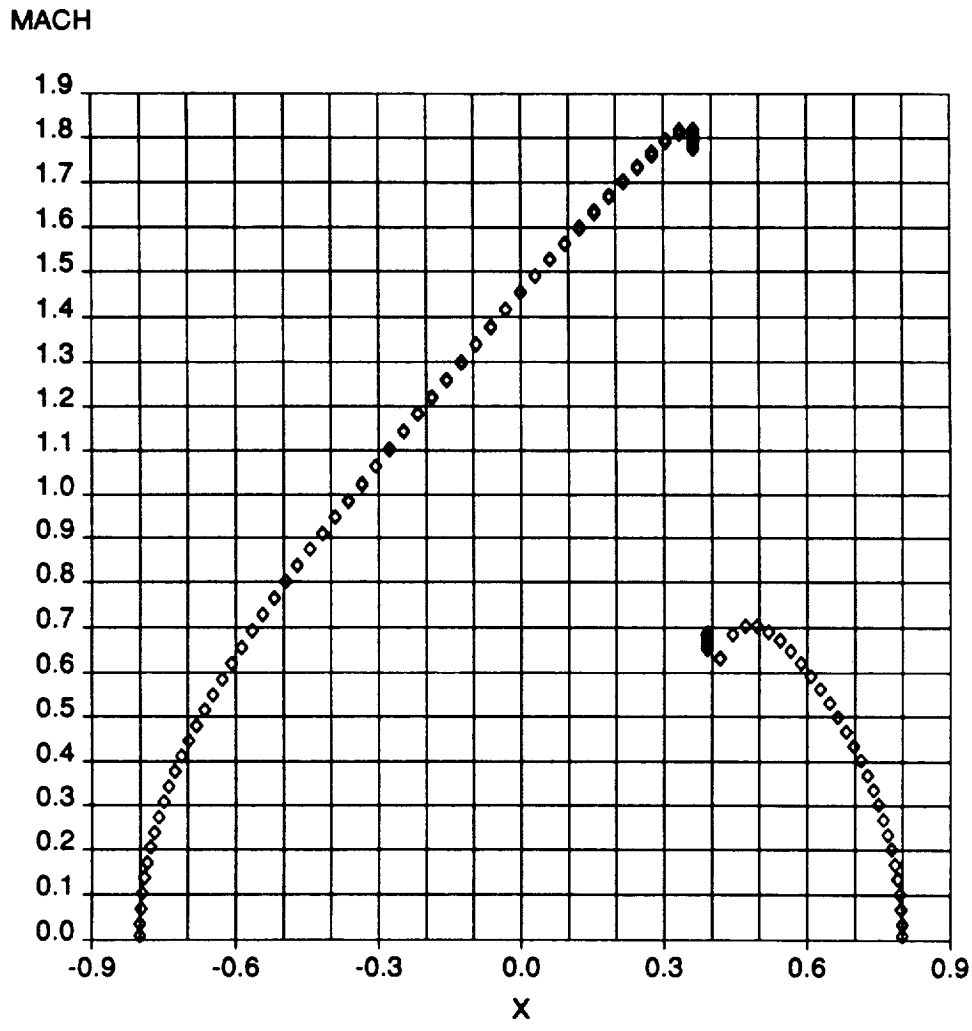


Figure 3.12: Surface Mach Numbers for Sphere, $M_\infty = 0.7$.

3.2.2 ONERA M6 Wing

The ONERA M6 wing was analyzed at $M_\infty = 0.84$ and $\alpha = 3.06°$ using grid sequencing. This a very popular test case for transonic flow codes and exhibits an oblique (supersonic to supersonic) shock as well as a normal (supersonic to subsonic) shock. Moreover, there is a fairly complicated shock pattern on the planform of the wing. Unless otherwise mentioned all the results in this sections were obtained using post processing.

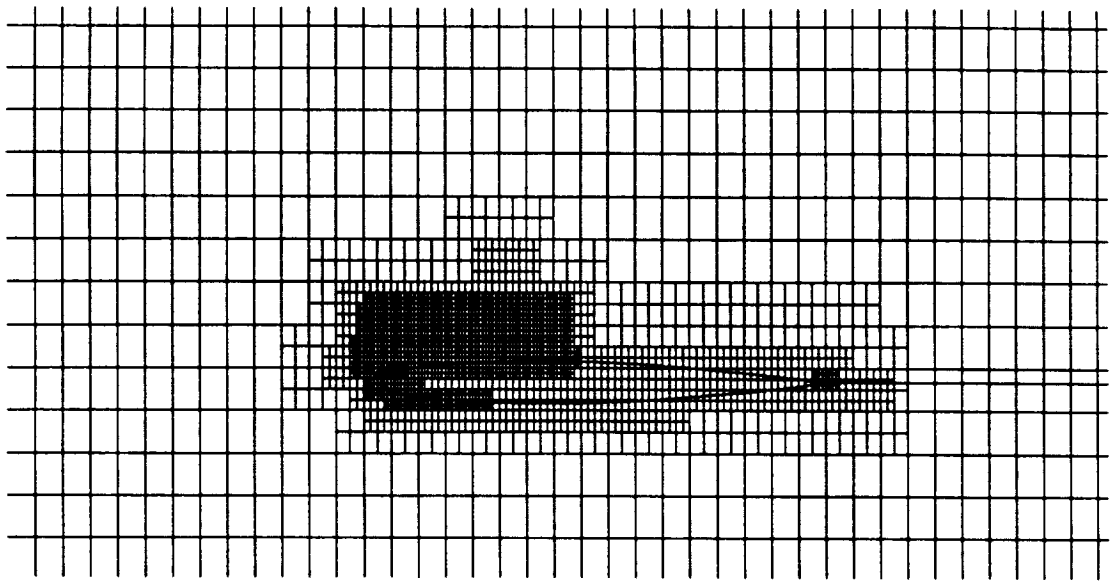
The TRANAIR results are compared to those obtained using the FLO28 code of Jameson [34]. The surface geometry used in the two solutions is identical. FLO28 solves the full potential equation using a surface fitted grid and is particularly well suited to simple wing geometries such as the ONERA M6 wing. The TRANAIR solution is obtained using a grid with about 311,000 elements whereas the FLO28 code solution was obtained on a grid with 364,000 cells. Dense grids were used in both codes to accurately capture the oblique shock. In Figure 3.13 two vertical cuts through the TRANAIR grid for this case are shown. Figure 3.14 is a waterline cut through the grid.

Figure 3.15 compares surface pressures at four stations with those obtained with FLO28. The TRANAIR solution was obtained using flux biasing and post processing. It is unclear in this case whether the second order dissipation FLO28 solution offers improved accuracy. In this problem, TRANAIR obtained comparable accuracy at comparable cost.

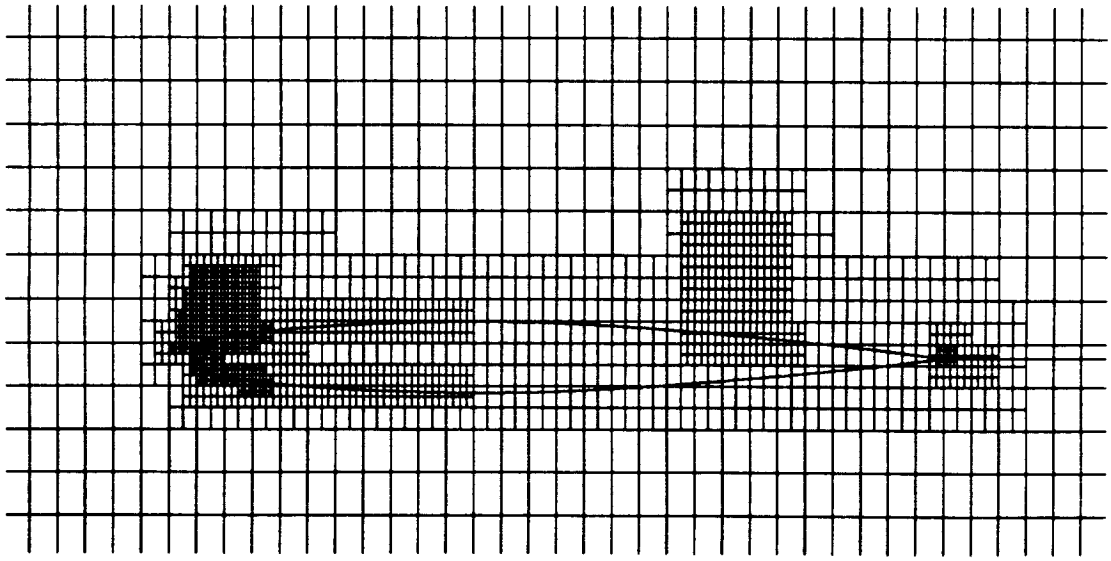
The ONERA M6 wing case was also analyzed using the solution adaptive grid feature. The initial grid had about 15,000 box elements. Intermediate and final target numbers of elements were specified as $N_I = 300,000$ and $N_F = 440,000$, respectively. A level limiting strategy was employed for the intermediate grids. The maximum level of refinement throughout the flow field was specified to be 4 levels below the global grid in all but the finest grid. In the finest grid 5 levels of refinement were permitted. One special region of interest was used to prevent refinement more than one level below the global grid in the tip region. No scaling of local error predictors was done.

In the resulting adaptive grid run, 6 grids were created. These contained approximately 39,000, 86,000, 184,000, 286,000, 312,000 and 423,000 elements. Figure 3.16 shows 70% span station cuts through the initial grid and final adaptive grid. Cuts through the final adaptive grid at 0% and 44% span are shown in figure 3.17.

Figure 3.18 displays computed pressure coefficients against percentage wing chord at 0% and 70% span for the initial grid, the second adaptive grid and the final adaptive grid. These results are generally quite accurate. One notices, however, that the oblique shock present on the wing at about 25% chord on the 70% span station is smeared. The oblique shock is a relatively weak phenomenon in this problem that can only be detected once a very fine grid is present.



90% Span



Plane of Symmetry (Root)

Figure 3.13: Two Cuts Through Grid for ONERA M6 Wing, $M_\infty = 0.84, \alpha = 3.06°$.

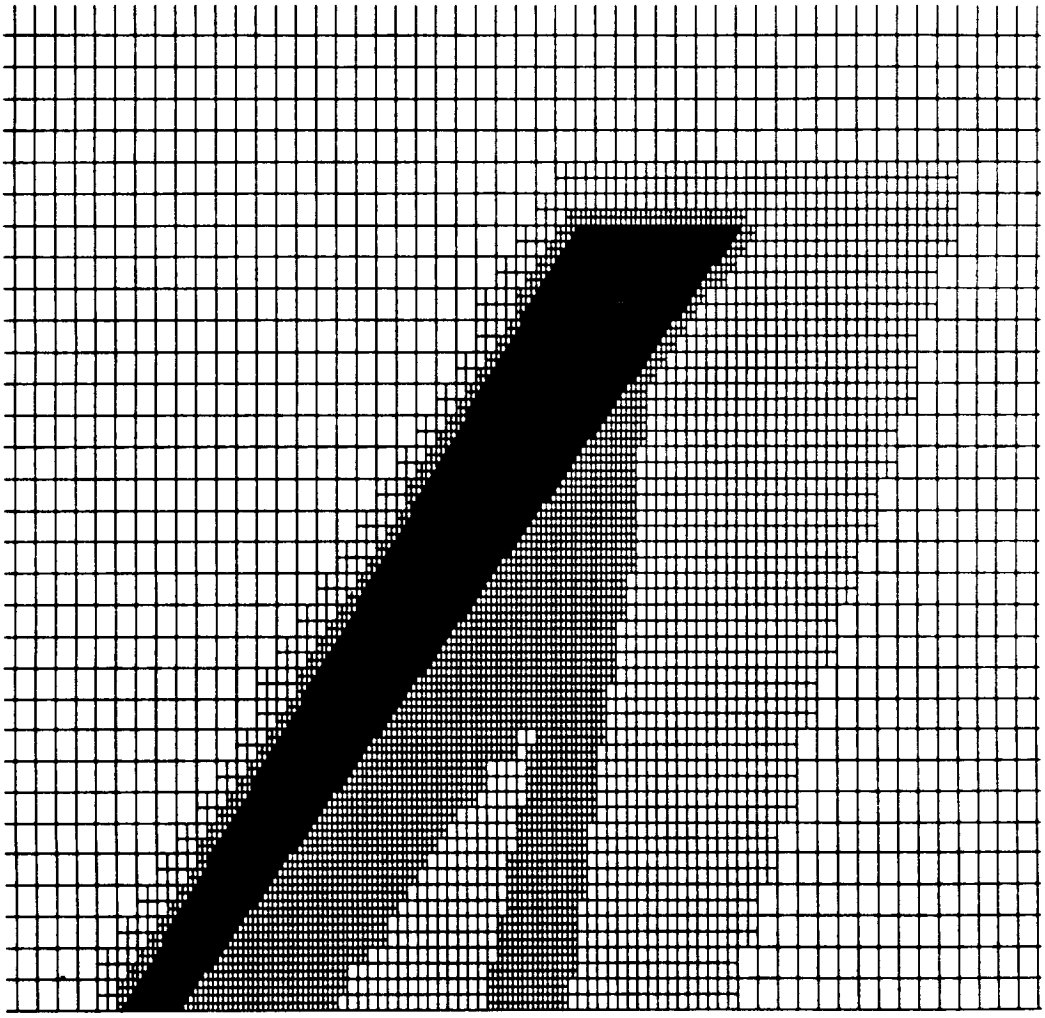


Figure 3.14: Waterline Cut Through Grid for ONERA M6 Wing, $M_\infty = 0.84, \alpha = 3.06°$.

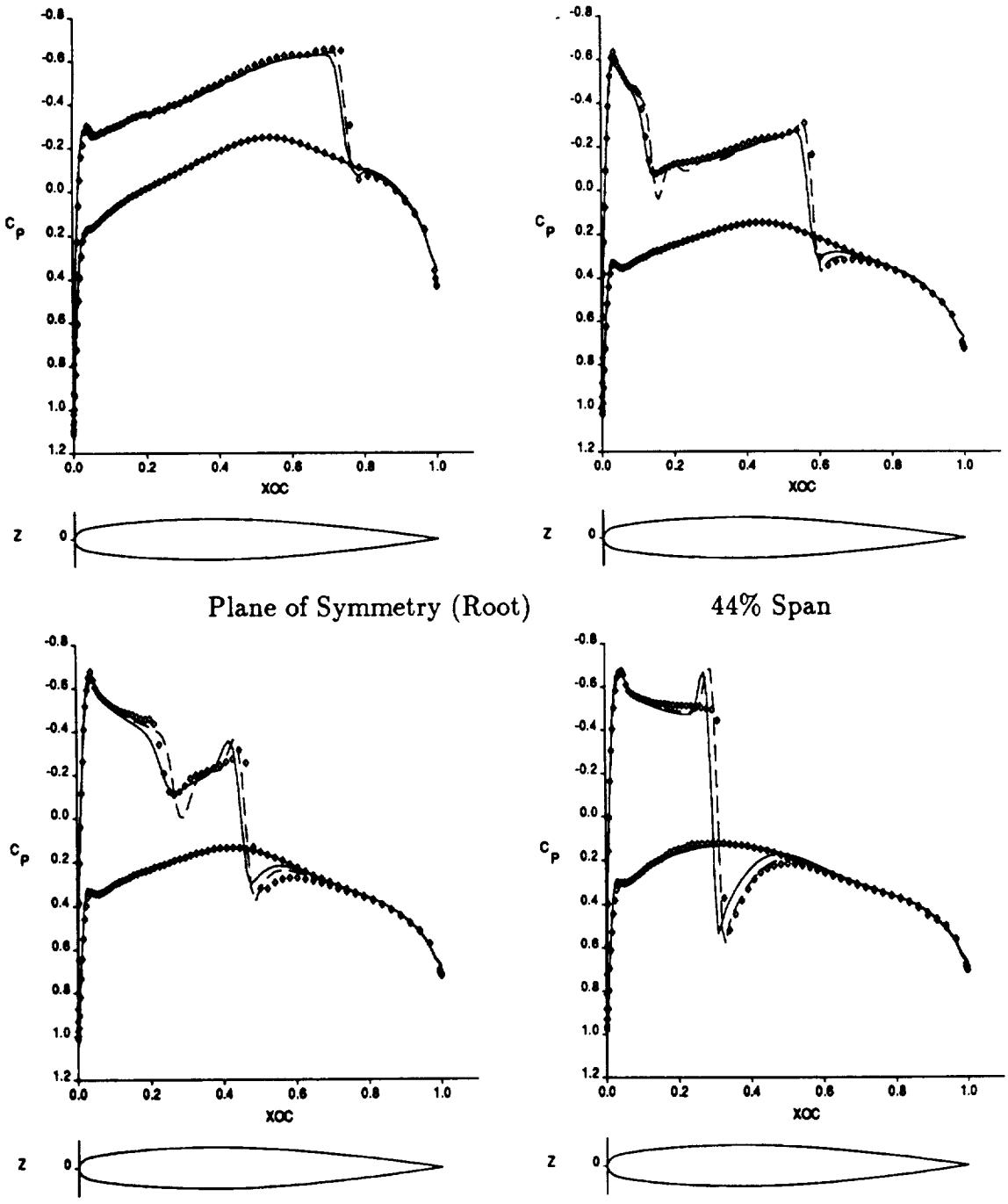
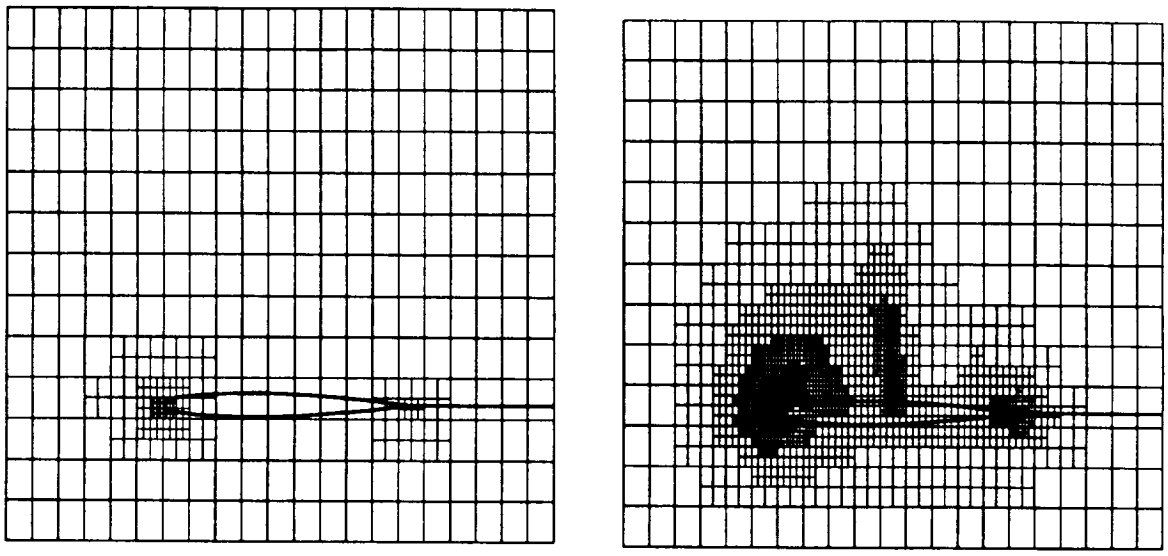


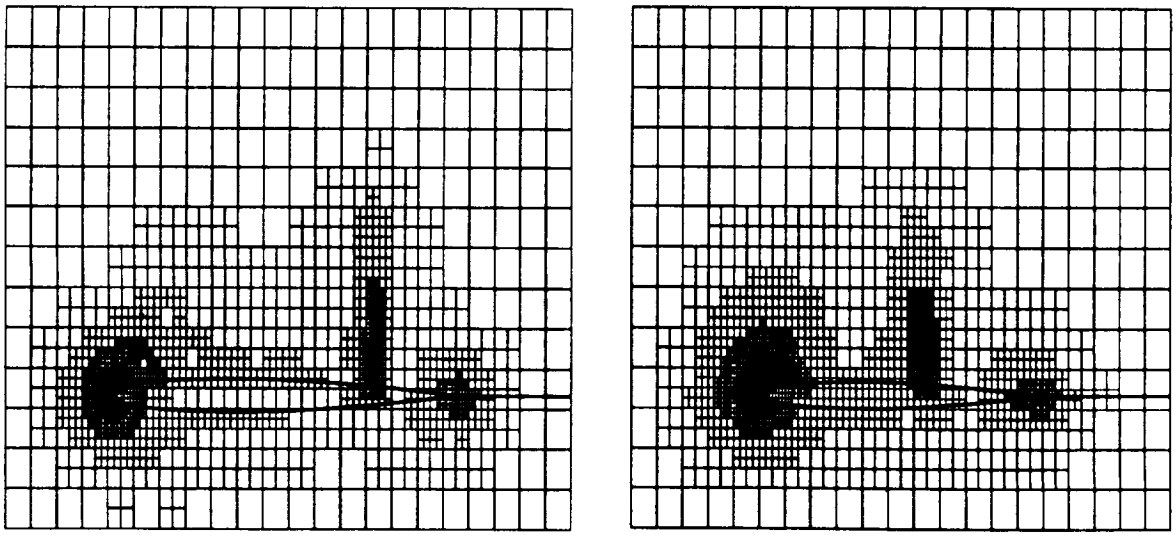
Figure 3.15: Comparison of Surface Pressure at Four Span Stations on ONERA M6 Wing, $M_\infty = 0.84, \alpha = 3.06°$.



70% Span, Initial Grid

70% Span, Final Adaptive Grid

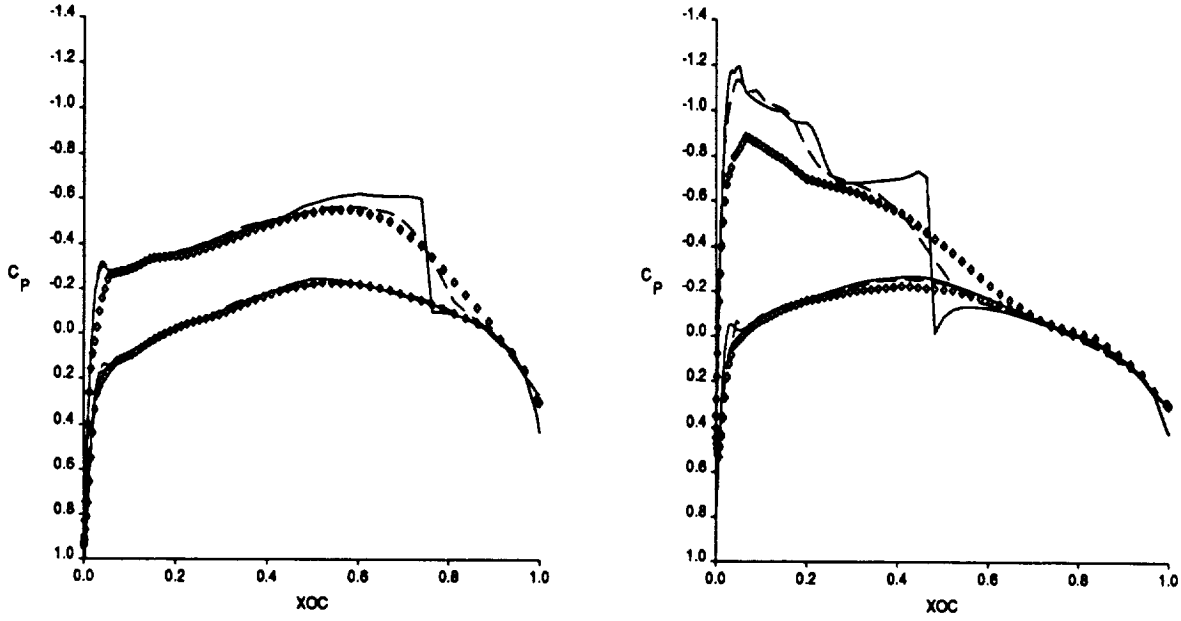
Figure 3.16: Grid Cuts at 70% Span for the ONERA M6 Wing, $M_\infty = .84, \alpha = 3.06°$.



0% Span, Final Adaptive Grid

44% Span, Final Adaptive Grid

Figure 3.17: Grid Cuts at 0% and 44% Span for the ONERA M6 Wing, $M_\infty = .84, \alpha = 3.06°$.



0% Span 70% Span

◇ Initial Grid Solution — — Second Adaptive Grid Solution
—— Final Adaptive Grid Solution

Figure 3.18: Pressure Coefficients for ONERA M6 Wing, $M_\infty = .84, \alpha = 3.06°$.

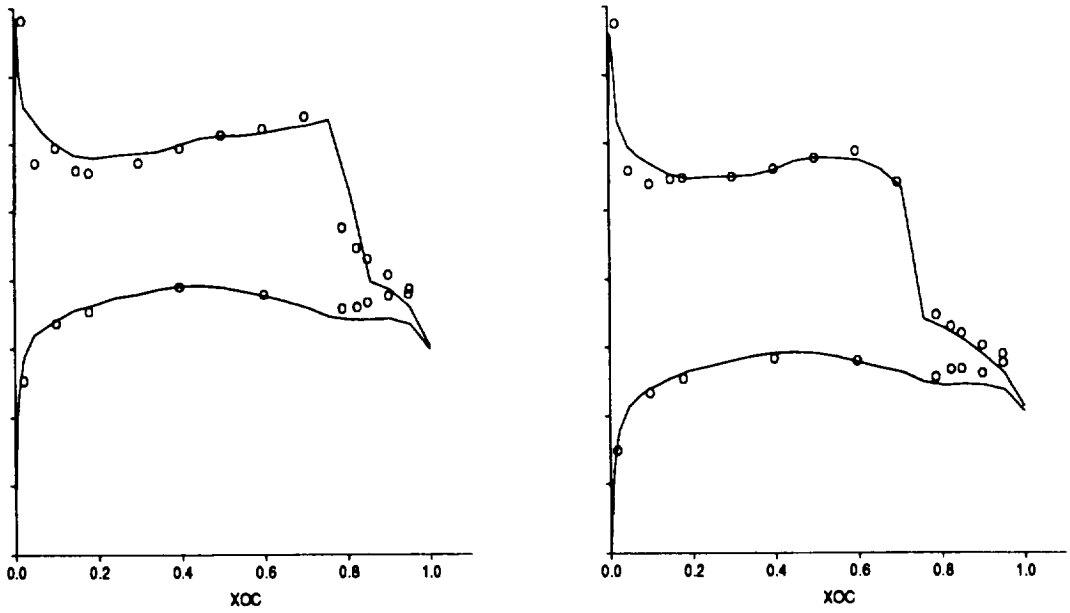


Figure 3.19: Wing Pressures for the F16, $M_\infty = 0.9, \alpha = 4.0°$.

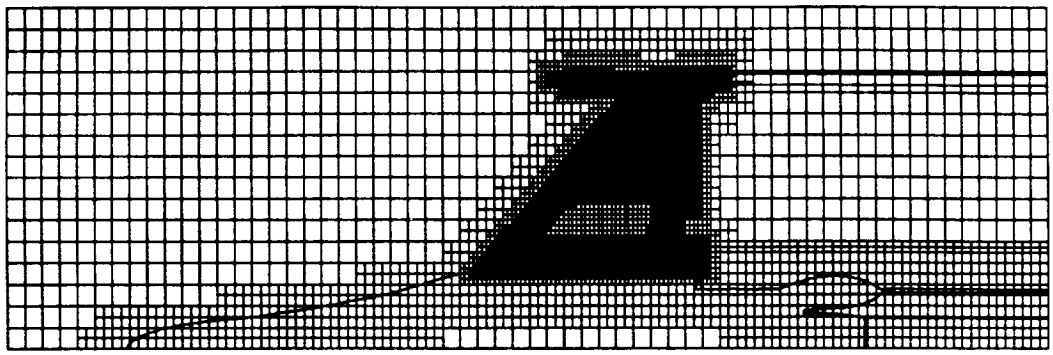
3.2.3 F16 Fighter Aircraft

The F16 shown in Figure 3.8 was analyzed at $M_\infty = 0.9$ and $\alpha = 4.0°$. The TRANAIR grid had about 189,000 elements. A comparison of computed surface pressure with wind tunnel data at two wing stations is shown in Figure 3.19. The agreement is good considering the fact that boundary layer effects are not yet included in TRANAIR.

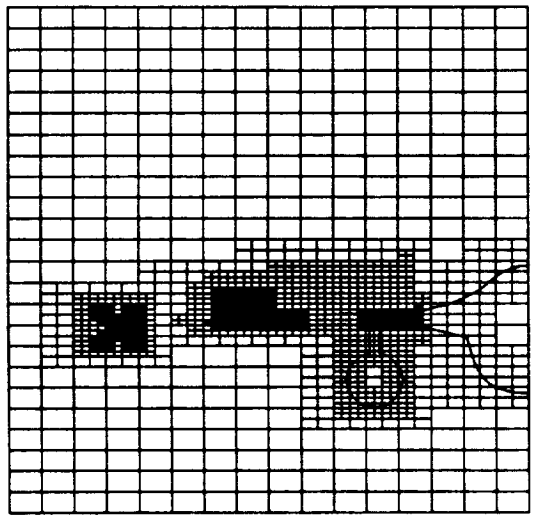
Another configuration of interest is the F16 with tanks and missiles shown in Figure 1.2. This is a very difficult case for surface fitted grid codes. The finest TRANAIR grid in this grid sequencing run contained about 216,000 elements. Figure 3.20 shows three plane cuts through the computational grid. Figure 3.21 compares computed surface pressure just inboard and just outboard of the tank strut with TRANAIR results for the F16 without tanks and missiles. The effect of the tank is as expected.

The F16 fighter with tanks and missiles was also analyzed using the solution adaptive method. This case involves extremely complex geometry leading to some very severe flow regions which provides a tough case for solution adaptive refinement. In the run a starting grid with 8,224 boxes (a global grid 33 x 9 x 13 with a maximum of 3 levels of panel induced refinement) was used.

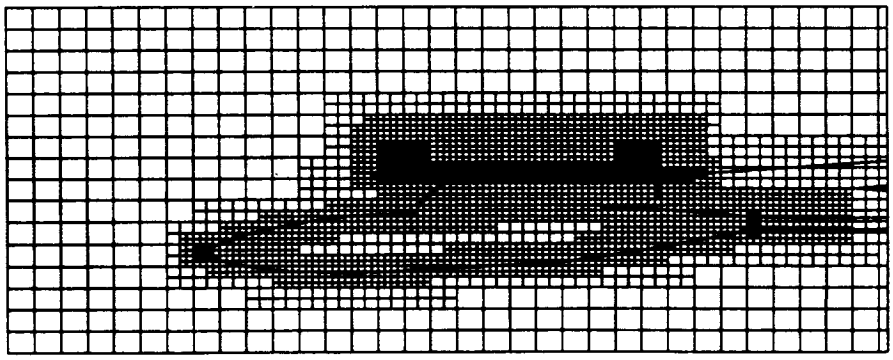
It was assumed that the region of primary interest was the wing/tank. Also the major features of the flow about the fuselage should be captured. Earlier test runs indicated several regions where developing high flow gradients necessitated restriction of refinement. In the wake region behind the missile, refinement was restricted to one



Waterline Cut Through wing

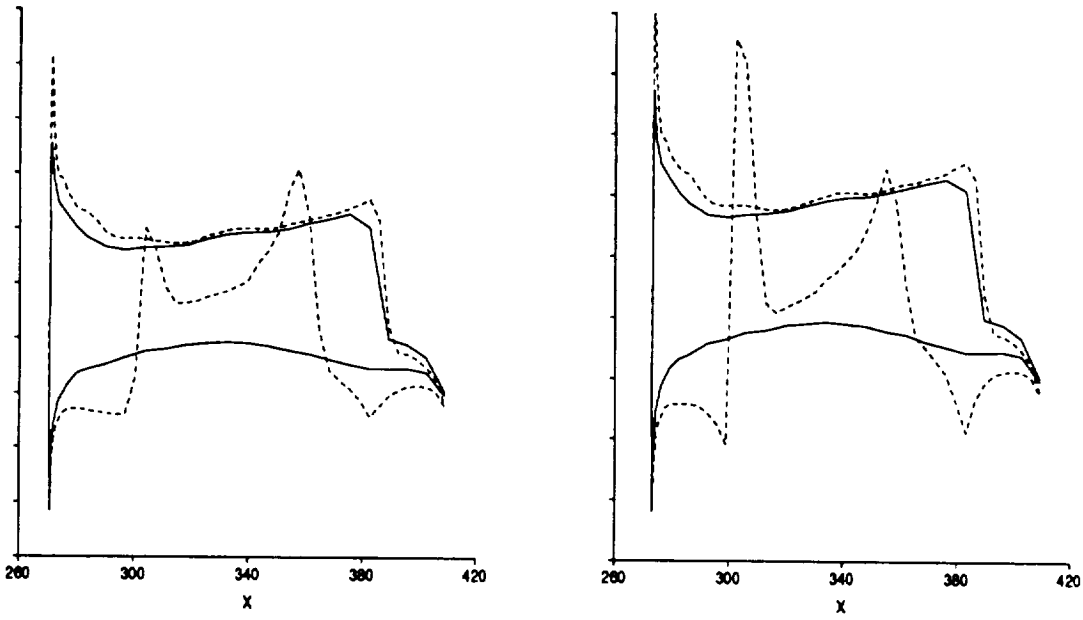


Vertical Cut



Cut Through Underwing Tank and Strut

Figure 3.20: Cuts Through the Grid for the F16 With Tanks and Missiles, $M_\infty = 0.9, \alpha = 4.0°$.



Inboard Side of Strut

Outboard Side of Strut

—— F16 Without Tanks and Missiles

– – F16 With Tanks and Missiles

Figure 3.21: Computed Wing Pressures for the F16 with Tanks and Missiles, $M_\infty = 0.9, \alpha = 4.0°$.

C-2.

level, also refinement was restricted to four levels on the fuselage and on the tank. A strong error indicator de-emphasis was necessary in the rear of the fuselage as fine grid in the cut-out region leads to very high flow gradients (and hence more grid refinement.)

Three adaptive grids were used, generating grids with 25814, 82492 and 256667 boxes. The final grid has a maximum of six levels of grid refinement, found at the wing leading edge. Fourth- and fifth-level grid is found in the shock regions, strut and tail leading edge, missile and the front of the fuselage. Fig. 3.22 shows cuts through the grid at $y = 0$ (plane of symmetry) , $y = 72$ (tank-strut location) and $z = 94$ (waterline cut through the wing).

Fig. 3.23 shows Cp at $y = 72$ (inboard side of the strut), and along the crown line of the front of the fuselage (the grid is shown in fig. 3.22). It is worth noting that when using adaptive refinement extra care must be taken with geometry definition. Figure. 3.24 shows the grid cut at $y = 48$ with an enlargement in the strake area. A small depression in the surface geometry is picked up by the error estimator and the grid is refined there to the maximum level.

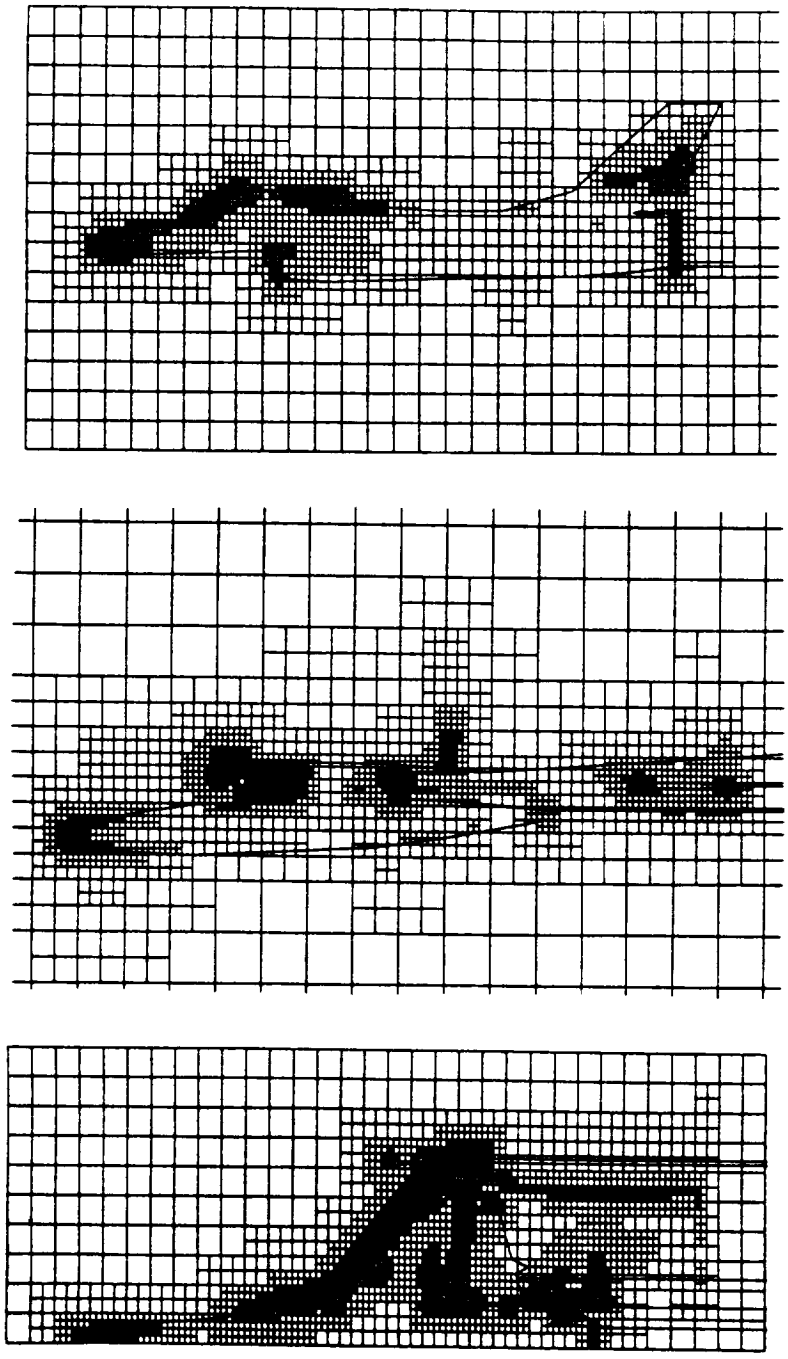


Figure 3.22: Cuts Through the Final Adaptive Grid for the F16 With Tanks and Missiles, $M_\infty = 0.9, \alpha = 4.0°$.

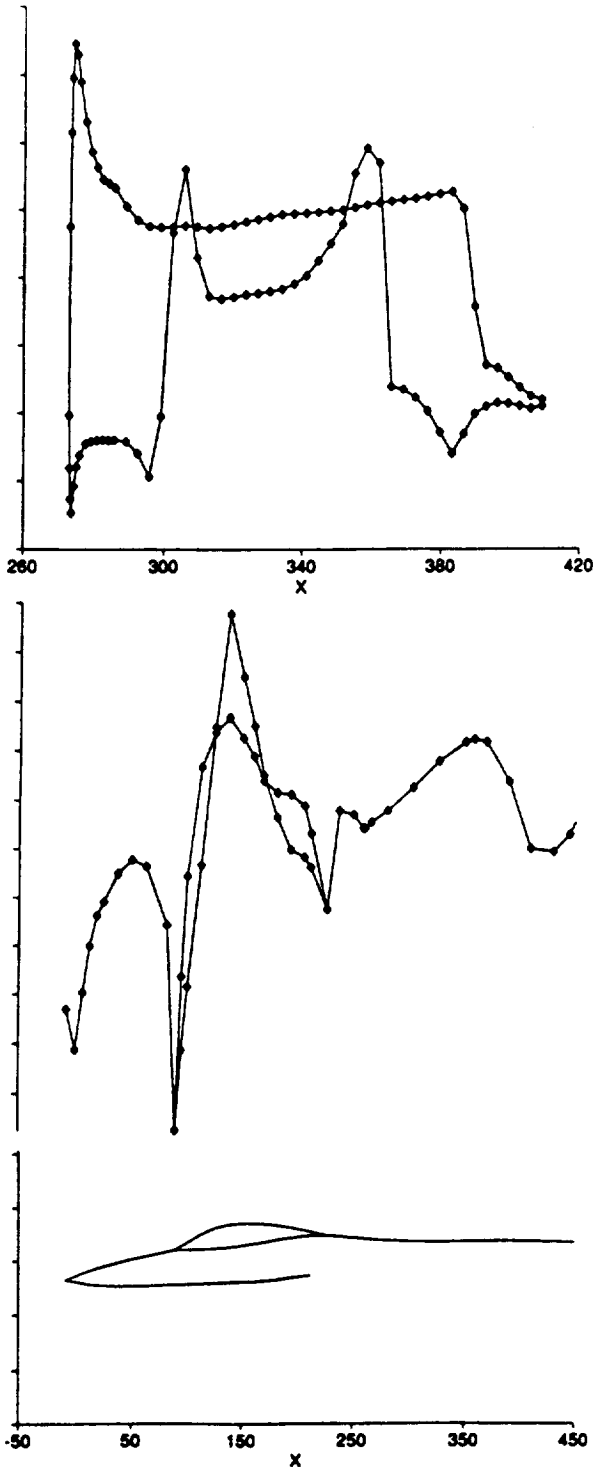


Figure 3.23: Computed Wing Pressures for Adaptive Grid Run of the F16 with Tanks and Missiles, Inboard Side of Strut and Crown Line, $M_\infty = 0.9, \alpha = 4.0°$.

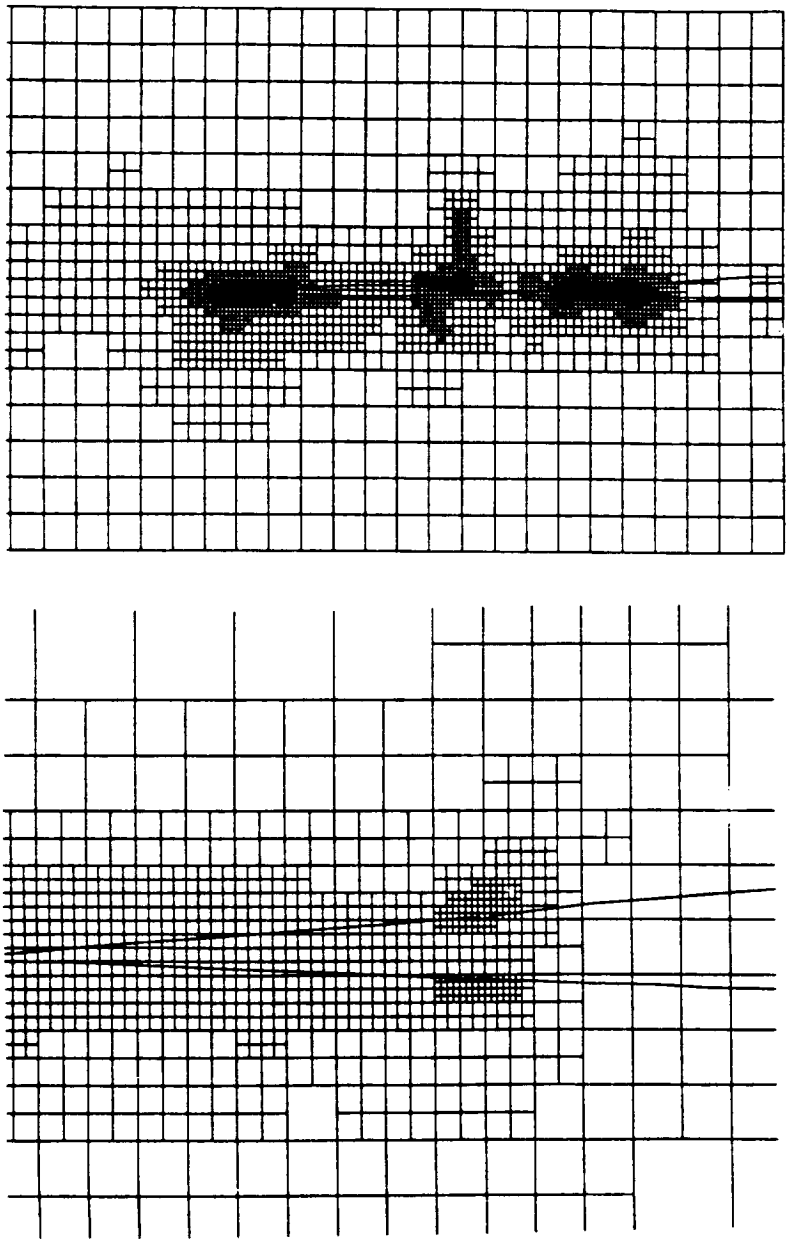


Figure 3.24: Cuts Through Adaptive Grid for the F16 With Tanks and Missiles near the Strake, $M_\infty = 0.9, \alpha = 4.0°$.

3.2.4 Boeing 747-200

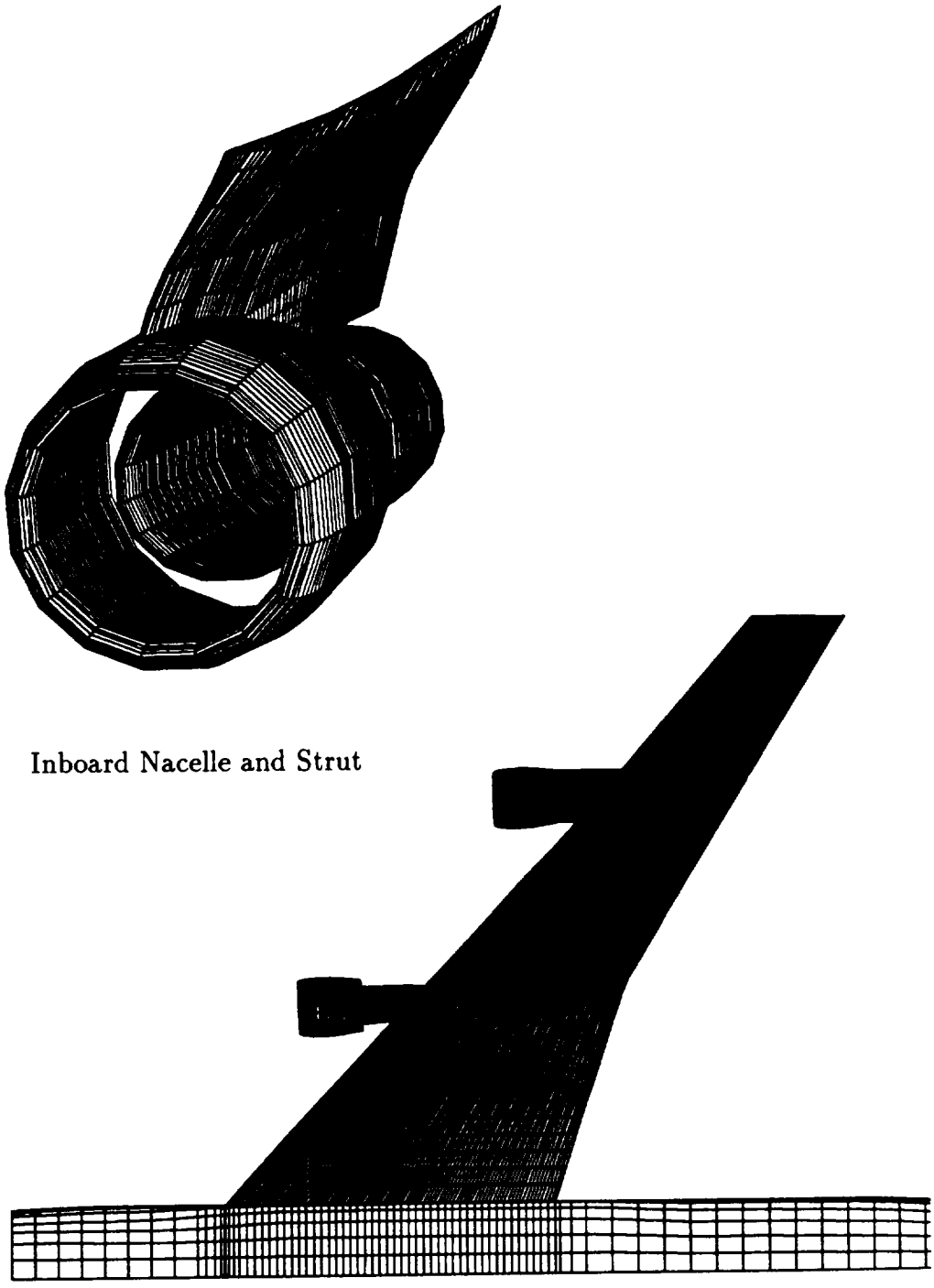
TRANAIR was used to analyze the flow about a 747-200 transport at $M_\infty = 0.8$ and $\alpha = 2.7°$. For this case, $M_\infty = 0.8$ is approximately the largest free stream Mach number at which an inviscid solver can obtain accurate results. The configuration included wing, body, struts and nacelles. Over 20,000 panels were used to describe the surface geometry of the symmetric model (see Figure 3.25).

The finest grid used in the grid sequencing run for this case consisted of approximately 219,000 finite elements. Figure 3.26 shows two cuts through the grid. The cut shown in Figure 3.26A is a yz plane cut and passes through the outboard nacelle strut and core cowl and through the prescribed wakes behind the inboard strut and nacelle. The cut shown in Figure 3.26B is an xz plane cut through the outboard nacelle.

Figure 3.27 compares TRANAIR results with wind tunnel pressure data at four span stations of the wing. Overall, one sees very good agreement with experiment. Most of the differences are seen in the upper surface pressures and are attributable to viscous boundary layer effects not currently modeled in TRANAIR. By comparing lower surface pressure profiles, one can clearly see the effect of the outboard nacelle at the 69% span station, which is shown in Figure 3.26B. The very high speed local flow near the leading edge on the upper surface at this span station is due to the presence of a strut cap connected to the outboard strut and wing leading edge.

The same case was also analyzed using the adaptive method. In applying the solution adaptive grid method an initial grid containing about 26,000 elements was used. The specified intermediate and target numbers of box elements were 125,000 and 250,000, respectively. Three special regions of interest were specified to guide the adaptive grid method. These included one about the wing tip, one under the wing enclosing the nacelles, and one above the wing. Four levels of refinement were permitted for all elements except in the special region of interest above the wing. There, 4 to 6 levels were permitted from the body to the wing tip in all grids except the final grid, for which 5 to 7 levels of refinement were permitted. Since wind tunnel test data for comparison were only available on the wing, the importance of this region was (further) emphasized by specifying a scaling factor of 8 for the local error predictors in the special region of interest containing the wing. A scaling factor of 2 was used in the special region of interest containing the nacelles. Predictors in other regions were not scaled.

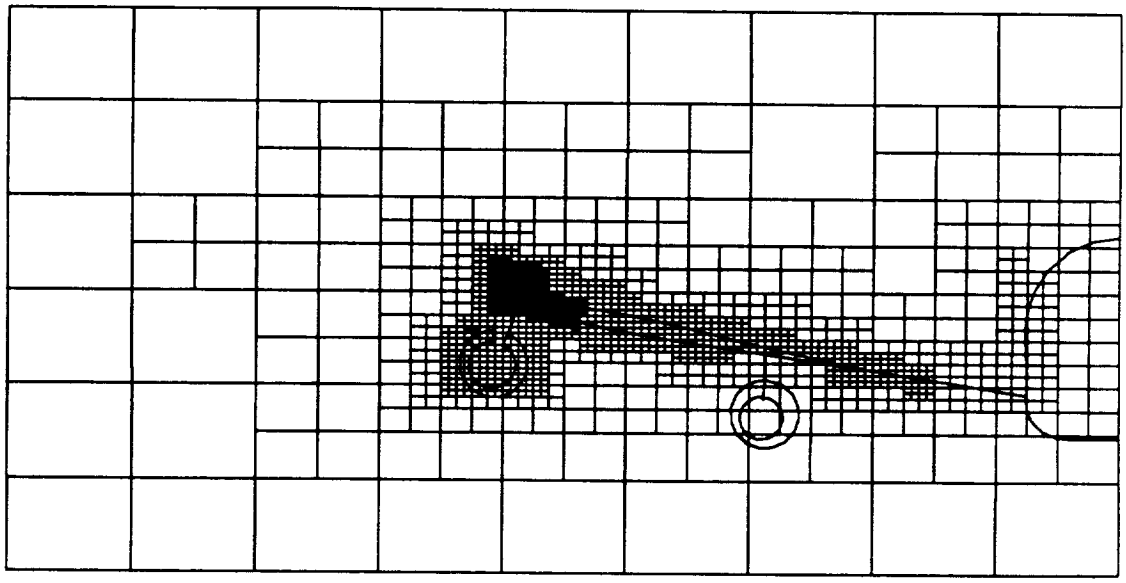
Four grids were created in a run with the solution adaptive grid method. These contained approximately 58,000, 123,000, 133,000 and 243,000 elements. Figure 3.28 shows 69% and 96% wing span station cuts through the initial grid and final adaptive grid. Figure 3.29 compares computed wing pressures with wind tunnel test data at four wing span stations. The method yielded results that compare favorably with the wind tunnel experimental data. In this case, the solution was actually computed *at* 69% span, resulting in an apparent difference from the results in Fig. 3.27 in which interpolation to 69% was performed.



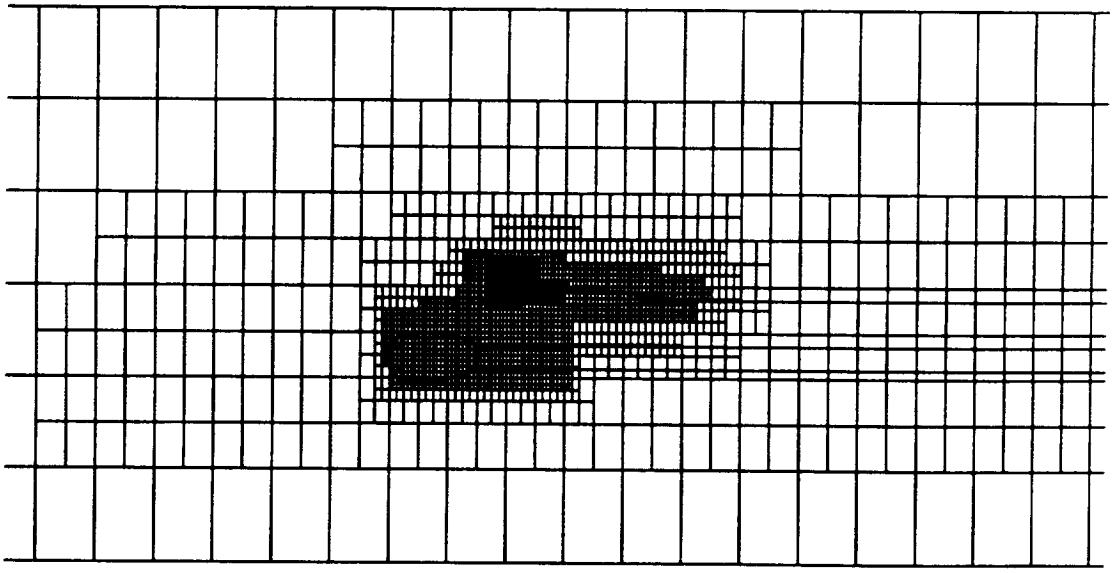
Inboard Nacelle and Strut

Top View of Wing and Body

Figure 3.25: 747-200 Transport Configuration.



A. Constant x Cut Behind Inboard Nacelle



B. Constant y Cut Through Outboard Nacelle

Figure 3.26: Two Cuts Through TRANAIR Grid for 747-200 Case.

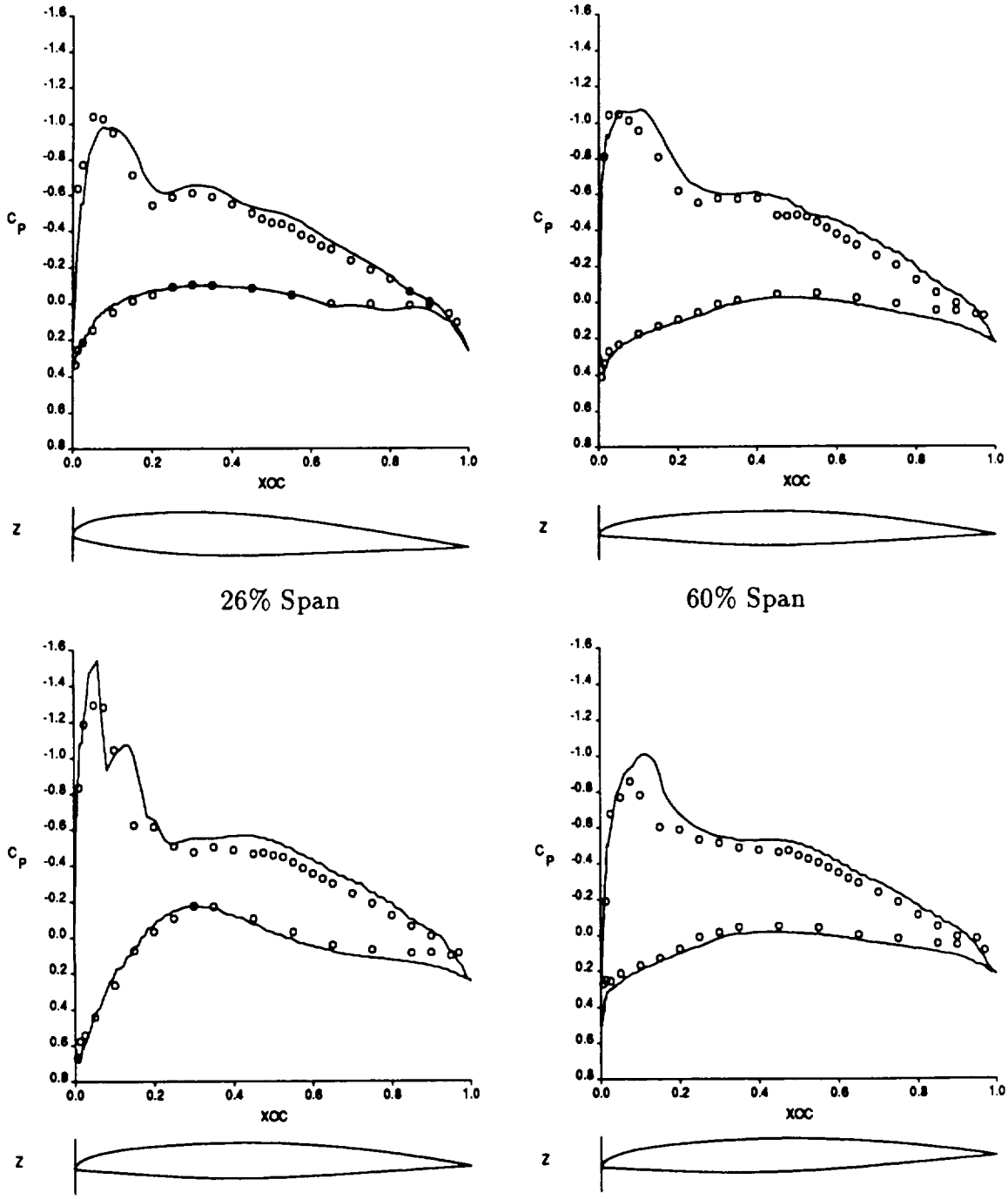
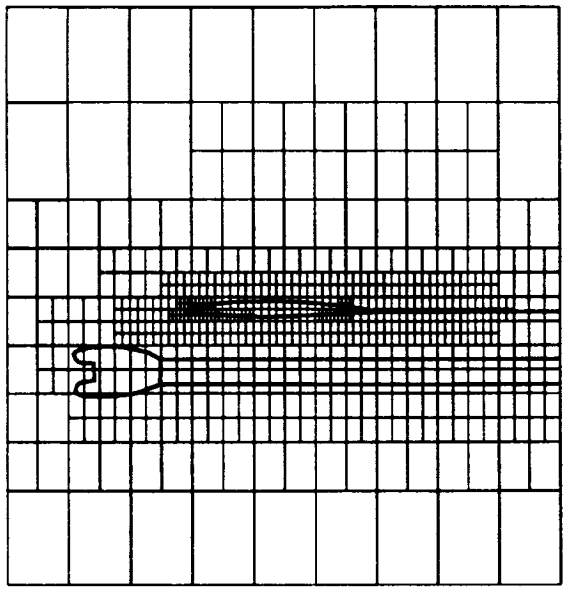
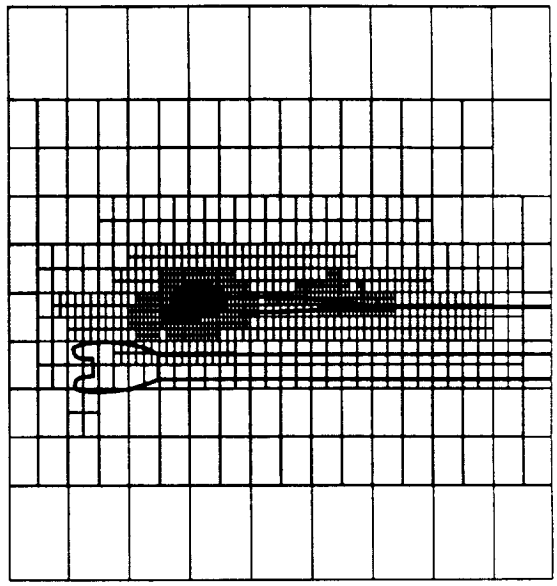


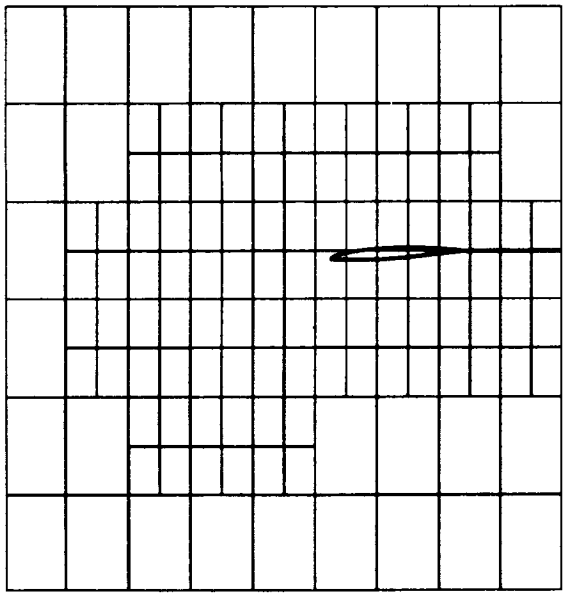
Figure 3.27: Wing Pressures for 747-200, $M_\infty = 0.8, \alpha = 2.7°$.



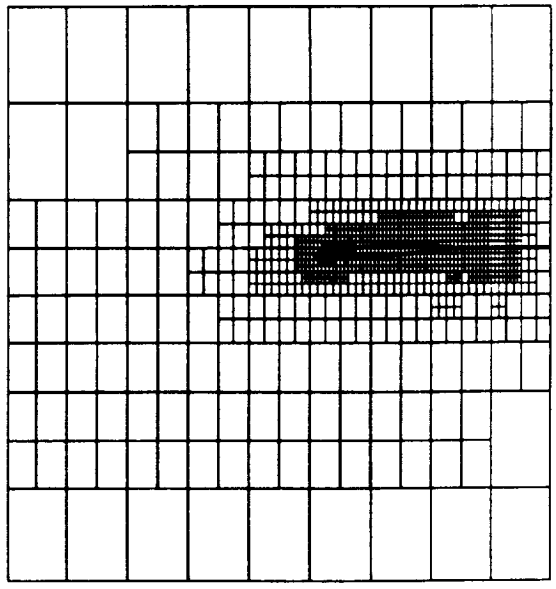
69% Span, Initial Grid



69% Span, Final Adaptive Grid

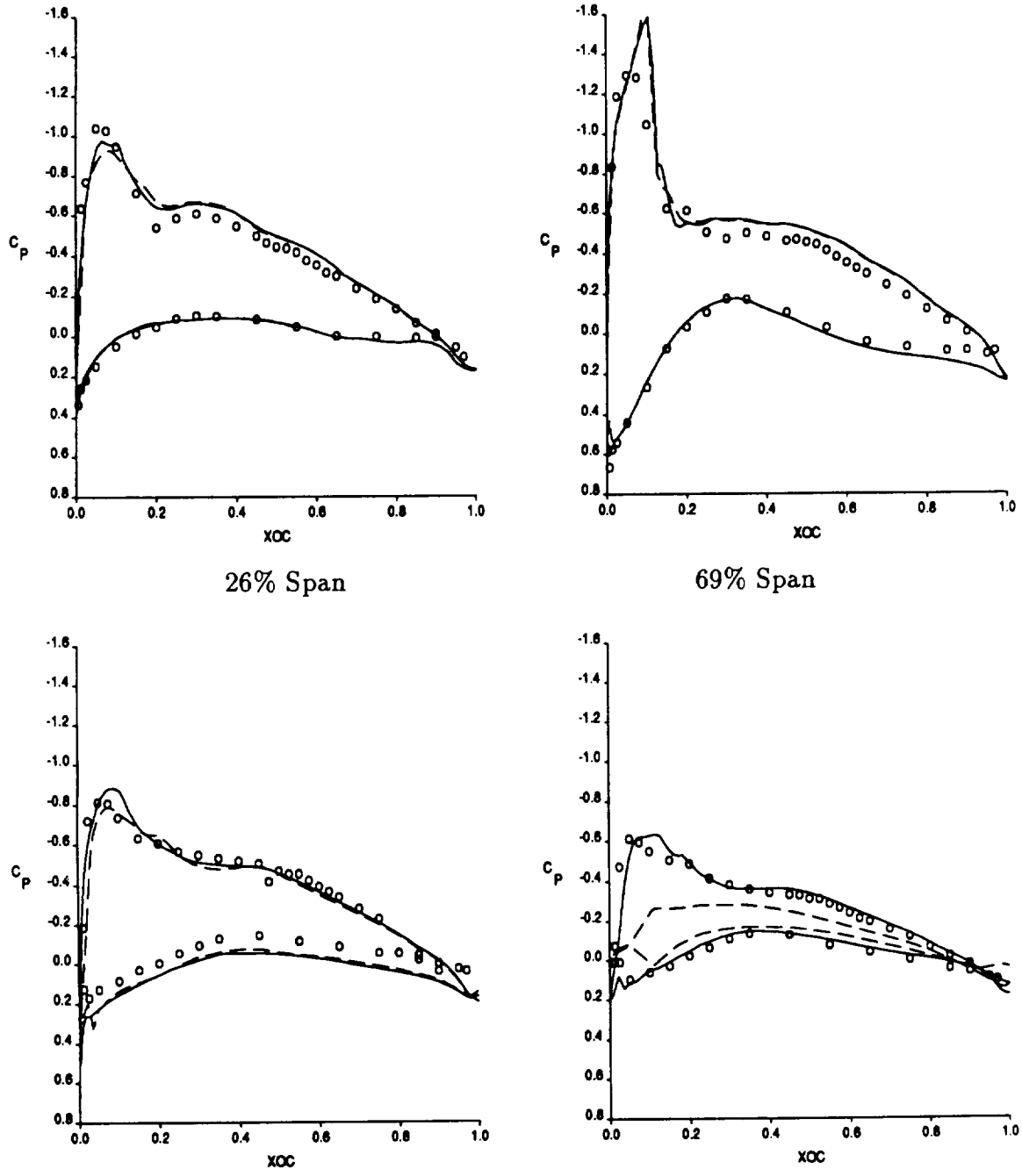


96% Span, Initial Grid



96% Span, Final Adaptive Grid

Figure 3.28: Grid Cuts at 69% and 96% Wing Span for 747-200, $M_\infty = .80, \alpha = 2.70°$.



26% Span

69% Span

89% Span

96% Span

— — Second Adaptive Grid Solution

—— Final Adaptive Grid Solution

o Wind Tunnel Test Data

Figure 3.29: Wing Pressures for 747-200, $M_\infty = .80, \alpha = 2.70°$.

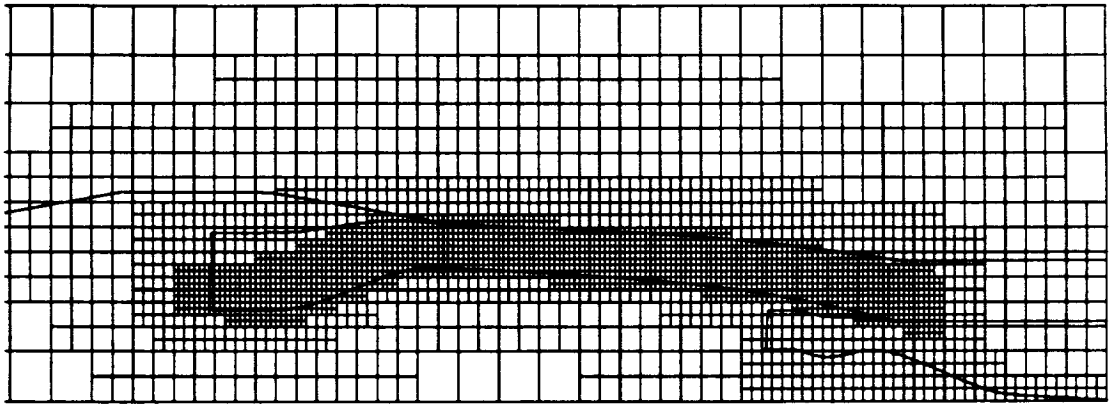


Figure 3.30: Cut Through Grid for an Axisymmetric Powered Nacelle, $M_\infty = 0.1$.

3.2.5 Axisymmetric Nacelle with Powered Plume

The next case illustrates the capability of the TRANAIR code to model different total pressure in an axisymmetric nacelle for which static test data is available [71]. The total pressure ratio in the powered stream was 2.807 and that in the primary stream was 2.3425. This configuration is shown in Figure 3.30 and had about 5000 panels. The wakes were paneled so that the powered streams maintain equal area downstream from the exits. Two planes of symmetry were used for this case. Results are shown for the final grid with about 85983 elements. Five grids were used in this grid sequencing run with 539, 880, 2828, 16030, and 85983 elements. Figure 3.31 gives the convergence history for this case. The steps on the coarser grids are scaled by the number of elements in the grid.

The static pressure on the core cowl is compared with experimental data and with the results of running a Navier-Stokes code PARC2D [71, 72] in Figure 3.32. In this case PARC2D predicted no total pressure loss in the fan stream. Thus, isentropic modeling can capture the major features of this flow.

3.2.6 Analysis of an Installed Transport with Power Effects

Finally, the analysis of a transport aircraft with installed powered nacelles is presented. The plumes behind the nacelle are simulated as regions of different total pressure and temperature. In Figure 3.33a and 3.33b, the paneling for the configuration and a typical section of the grid with about 230,000 boxes are shown. In Figure 3.33c and 3.33d the pressure computed at an underwing station and inboard strut station with and without power (flight idle (ram) and cruise conditions) are compared. The effect of power on the local flow is obvious. This case demonstrates the capability of the TRANAIR code to handle power effects, a capability usually associated with an Euler formulation.

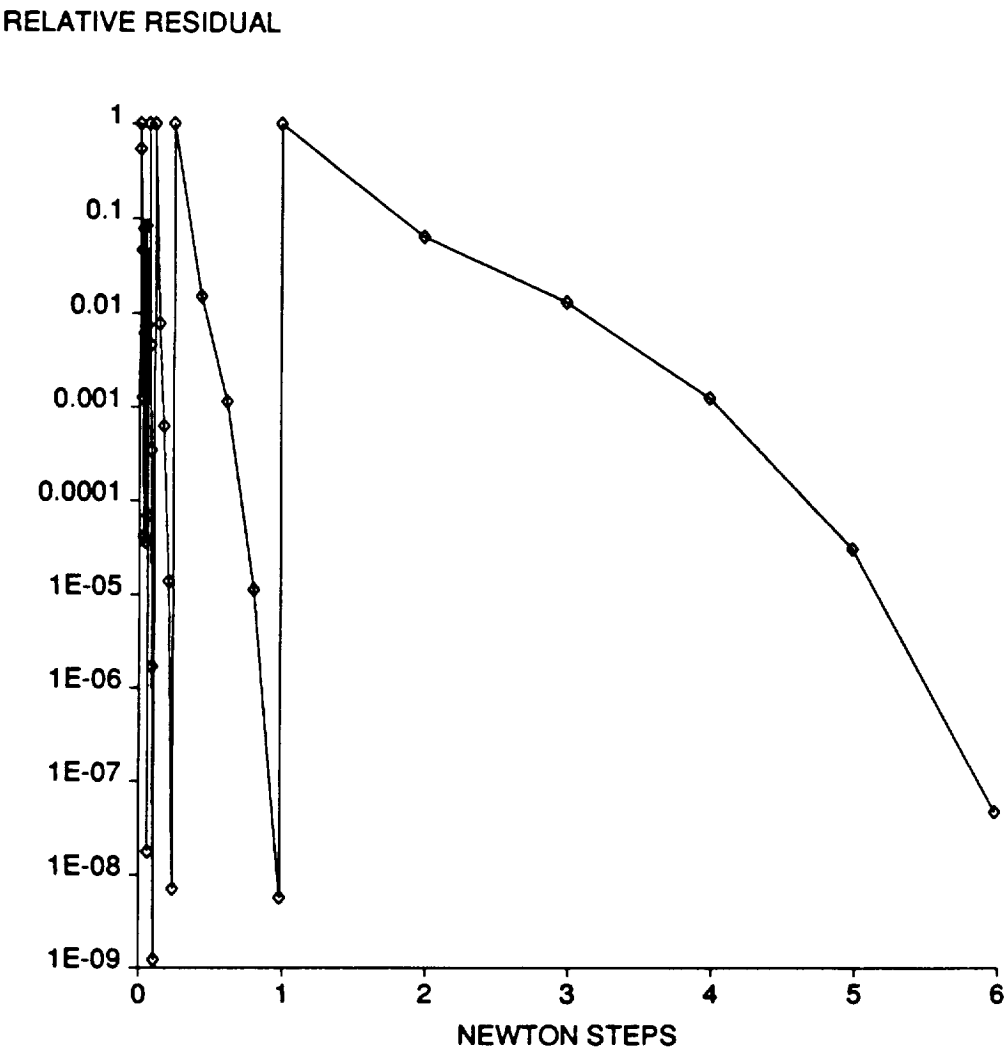


Figure 3.31: Convergence History for Grid Sequencing Method for Axisymmetric Powered Nacelle Case, $M_\infty = 0.1$.

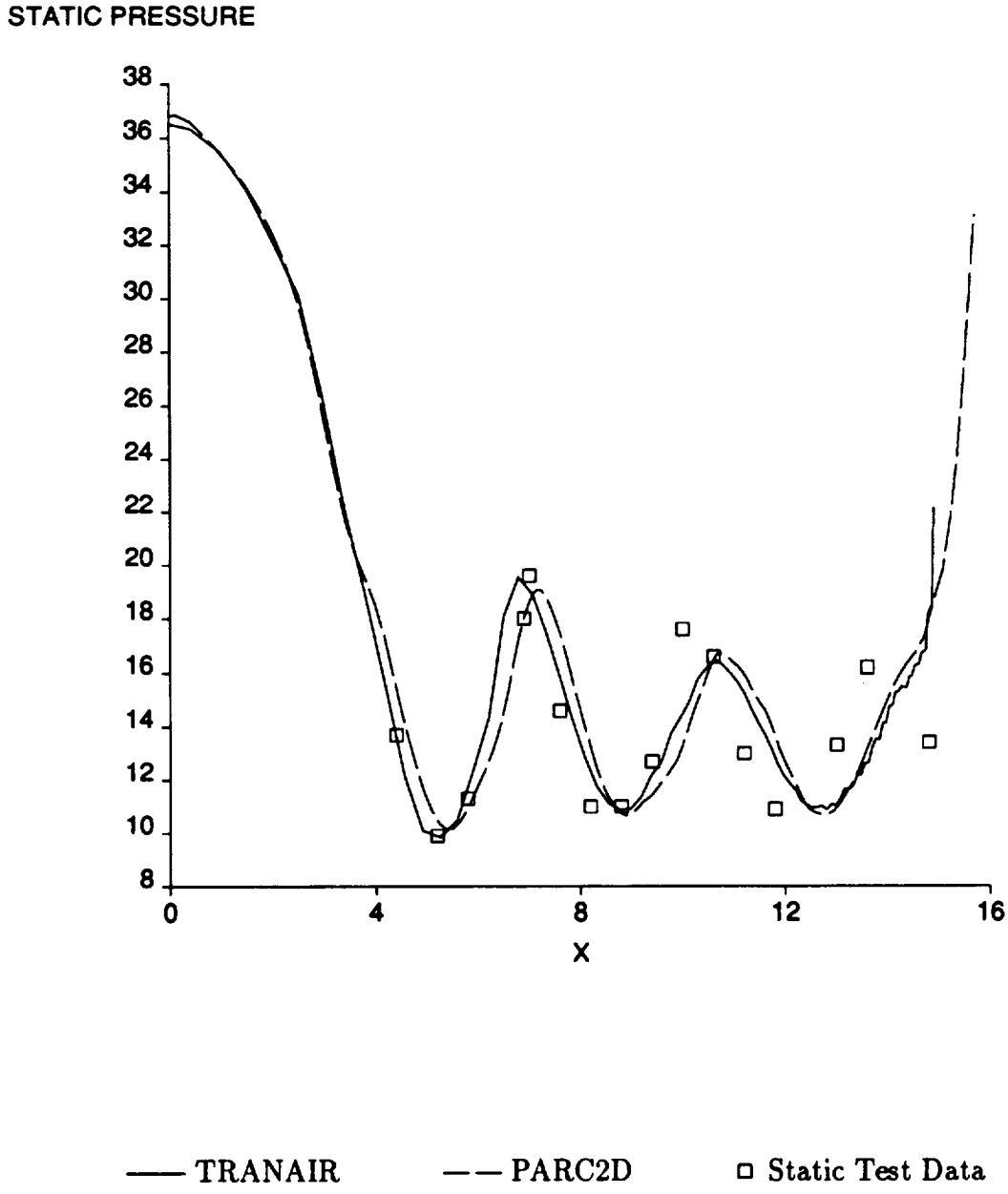
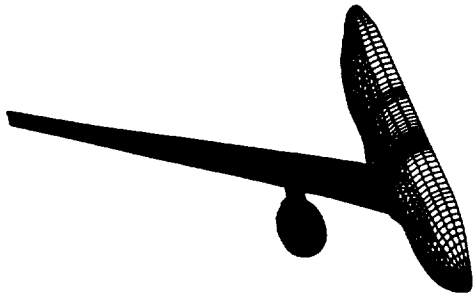
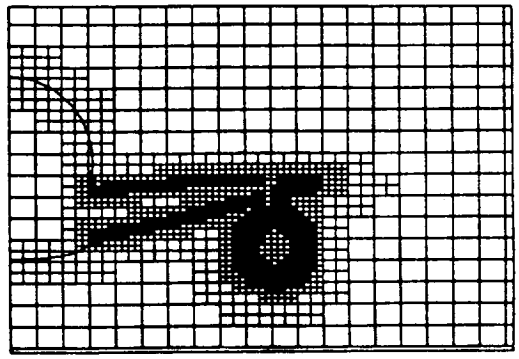


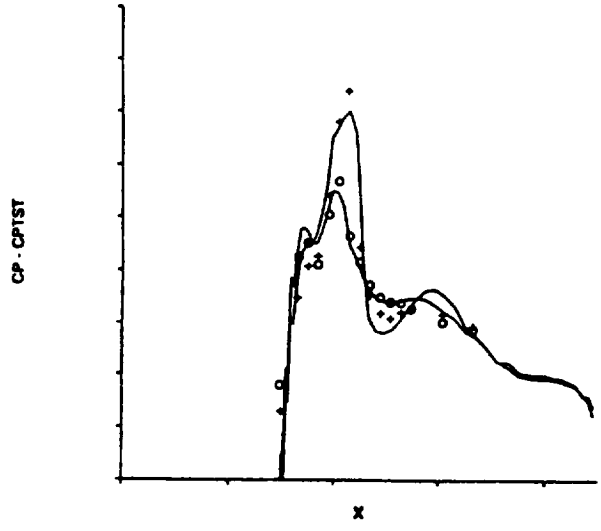
Figure 3.32: Static Pressure for Axisymmetric Powered Nacelle Compared to Experiment and Navier-Stokes Code Results, $M_\infty = 0.1$.



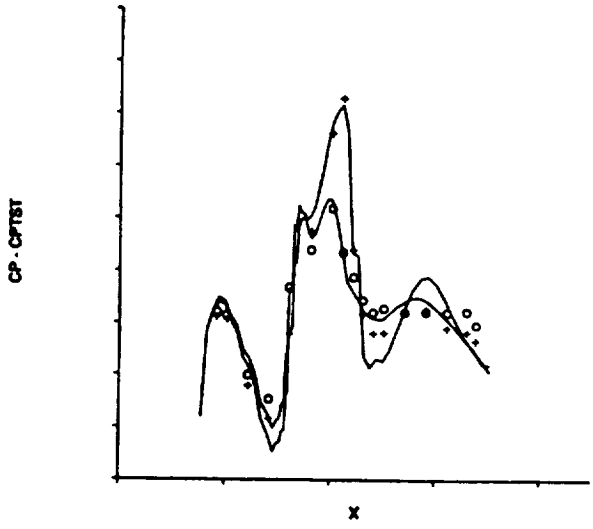
a) Surface Paneling



b) Section of TRANAIR Grid



c) Underwing Station



d) Inboard Strut Station

o Ram Test Data
\+ Cruise Test Data

—— TRANAIR
—— TRANAIR

Figure 3.33: TRANAIR Analysis of a Transport with Wing/Body/Nacelle/Strut and Power.

3.3 RESULTS FOR SUPERSONIC FREE STREAM FLOW

The supersonic free stream capability has been added to TRANAIR only recently and has not been exercised as thoroughly as the subsonic free stream capability. The supersonic free stream capabilities of TRANAIR have been tested on a cone-sphere configuration (compared with an analytic solution), two delta wing configurations with supersonic and subsonic leading edges, respectively (compared against a linear panel code solution and against the SIMP full potential code[73]), and an F16 configuration (comparison with experimental data and with a solution obtained with the SIMP full potential code). All configurations were analyzed with the solution adaptive gridding option of TRANAIR. In addition, to demonstrate the abilities of the solution adaptive gridding to capture a bow shock, a solution has been obtained on the reversed cone-sphere configuration – a sphere-cone. Some limited comparisons have been made with experimental data obtained from some standard textbooks.

3.3.1 Cone-Sphere Configuration.

Figure 3.34 compares the pressure distribution obtained with TRANAIR, with SIMP, with EMTAC[74], and with the analytic solution for flow over a cone of a 10 degree half angle. The flow conditions were a free stream Mach number $M_\infty = 1.414$ and $\alpha = \beta = 0°$.

The radius of the cone at the spherical cap was 1.76327 units. To avoid contamination of the surface solution by reflections of shocks from the outer faces of the global grid, the global grid was extended to twelve units normal to the axis of the cone. The final computational grid generated by TRANAIR is illustrated in Figure 3.35. Solution adaptation was performed five times. A minimum of one level of refinement near the boundary and a maximum of three levels was specified for the candidate initial grid.

Note that the inviscid solution (obtained by TRANAIR, SIMP and EMTAC) predicts a continued expansion to vacuum on the downstream end of the cone-sphere. In a real flow, viscous effects would produce separation before the expansion reached vacuum. The fictitious gas Mach number was raised to 7.0 for this analysis. Even at this value, the Mach number of the flow expands beyond Mach 15 before shocking down to stagnation at the aft portion of the sphere. The fictitious gas Mach number capability of TRANAIR allows it to obtain a solution to this case. It is interesting to note that with the fictitious gas model, the TRANAIR solution matches the Euler code solution obtained by EMTAC, rather than the SIMP solution. Because of the large velocity gradients in the flow field in the aft portion of the cone-sphere, all of the solution adapted gridding has been attracted to the downstream spherical cap. The tip "shock" went undetected by the solution adaptive gridding in the sense that no refinement was produced in the field to capture the tip discontinuity. However, the surface pressure distribution agrees with the analytic solution, probably because the tip shock is a weak phenomenon.

3.3.2 Delta Wing Configurations.

Figure 3.36 compares the spanwise pressure distribution for supersonic free stream flow past a delta wing with subsonic leading edges with a linear panel solution. The pressure distribution is illustrated at the 90% chord station. The free stream conditions were $M_\infty = 1.414$ and $\alpha = \beta = 0^o$. The thickness at full chord on the plane of symmetry was 2% of chord with a linear variation of thickness from the tip to full chord and in the direction normal to the plane of symmetry. The leading edge of the wing is swept to an angle of 60^o. Solution adaptive gridding with four cycles of grid adaptation were performed. Figure 3.37 illustrates representative cuts through the computational grid.

Figure 3.38 illustrates the pressure distribution on a supersonic leading edge delta wing. The sweep angle of the subsonic leading edge wing was changed to 30^o for this case. The TRANAIR solution agrees with the linear panel code solution up to the Mach cone emanating from the tip of the delta wing. The linear solution yields a constant Cp in the spanwise distribution, while the TRANAIR solution overshoots, presumably due to nonlinear effects. It is also possible that additional grid density might remove the overshoot. Because the flow gradients are relatively mild, some "hands-on" specification of grid may be required for this case. Figure 3.39 illustrates several cuts through the computational grid for the final (fourth) solution adapted grid obtained by the calculation.

3.3.3 F16 Configuration.

Figures 3.40 through 3.45 compare surface pressure distributions on the F16 configuration as predicted by TRANAIR and by SIMP. Figures 3.40 through 3.42 illustrate top, bottom and side views of the pressure distributions at free stream conditions of $M_\infty = 1.414$ and $\alpha = 4^o$. Figures 3.43 through 3.45 compare the solutions for free stream conditions of $M_\infty = 2.0$ and $\alpha = 2^o$. In general the correlation in the pressure distributions is quite close. The TRANAIR pressure predictions tend to be smoother and less jagged. At $M_\infty = 1.414$ the SIMP code produced an abnormal feature at the outboard trailing edge of the wing. This abnormality propagated in an inward direction toward the tail. When this intersects the configuration at the horizontal tail, a number of unusual reflections occur, contaminating the SIMP solution. The origin of this feature is not presently known.

Another significant difference at both $M_\infty = 1.414$ and $M_\infty = 2.0$ occurs on the lower surface in the region where the strake and body join. This is attributed to some differences in the gridding of the surface for SIMP. It was not possible to obtain an accurate representation of the diverter channel with SIMP and so the surface geometry was modified in this region. The result downstream of this modification was a more sudden change in slope in the region under the strake, resulting in a greater degree of stagnation in this area of the configuration. This is an artifact of the inability of SIMP to generate a grid which accurately describes the configuration geometry. TRANAIR accepts the paneled definition of the F16 geometry and hierarchically refines the grid where required by solution gradients to more accurately model the flow field.

Figure 3.46 compares the pressure distribution predicted by TRANAIR with wind tunnel measurements on an F16 configuration with tip missiles at $M_\infty = 1.2$ and $\alpha = 4^o$. Note that this configuration, (because of the existence of multiply connected regions in streamwise cuts of the configuration) could not be run in the SIMP code. Figure 3.47 illustrates some representative cuts of the final computational grid for the TRANAIR solution. Three cycles of grid adaptation were performed.

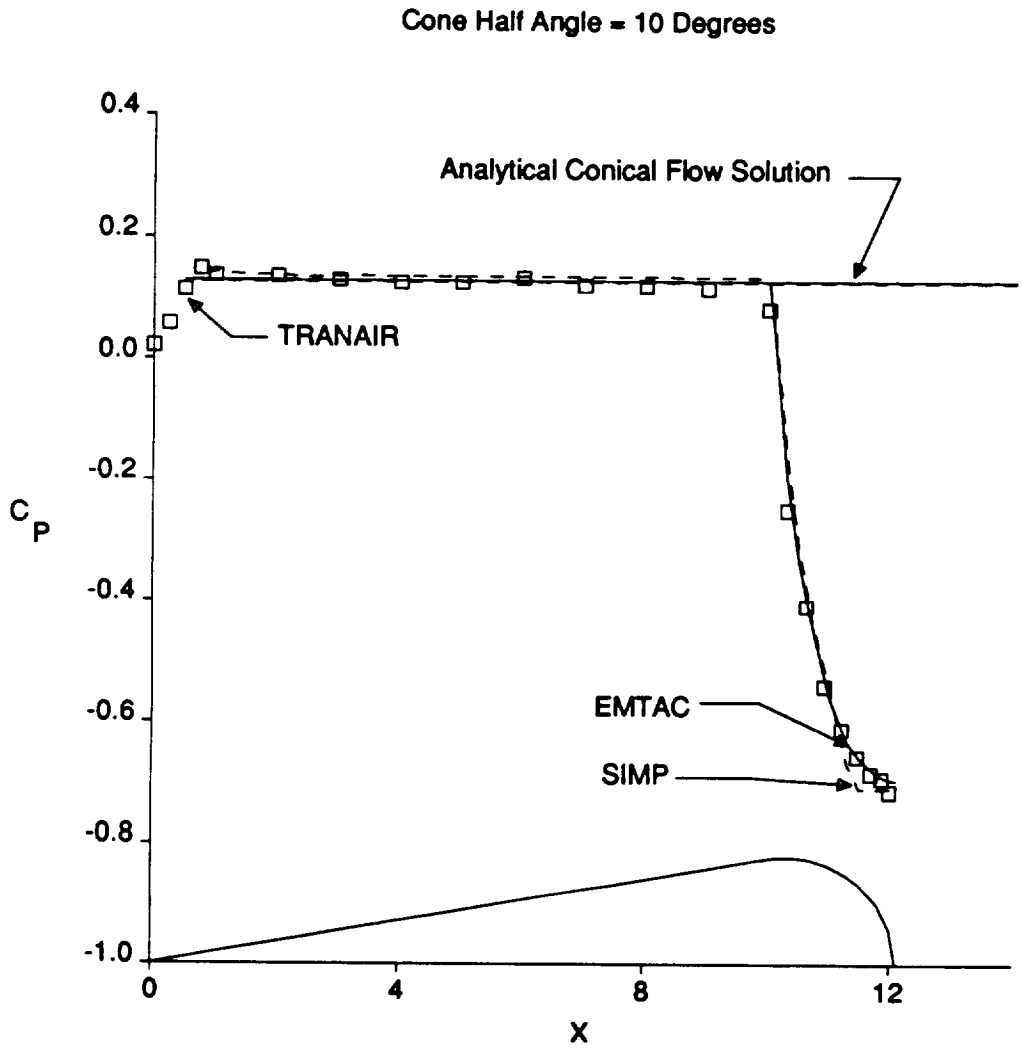


Figure 3.34: Cp Distribution on Cone Sphere at Mach 1.414. TRANAIR vs SIMP vs EMTAC and Analytic Solution.

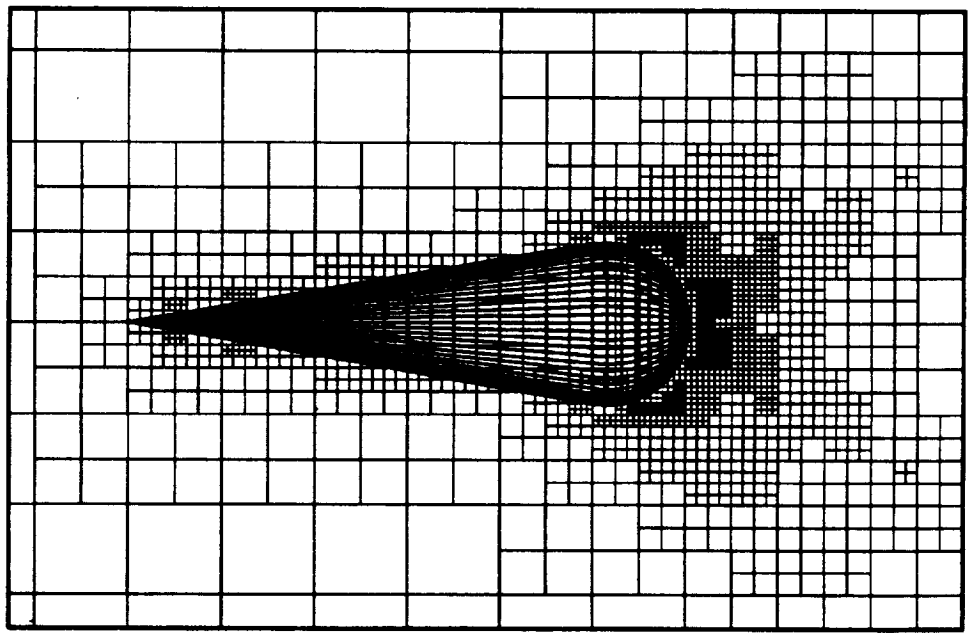


Figure 3.35: Final Computational Grid for Cone-Sphere Configuration.

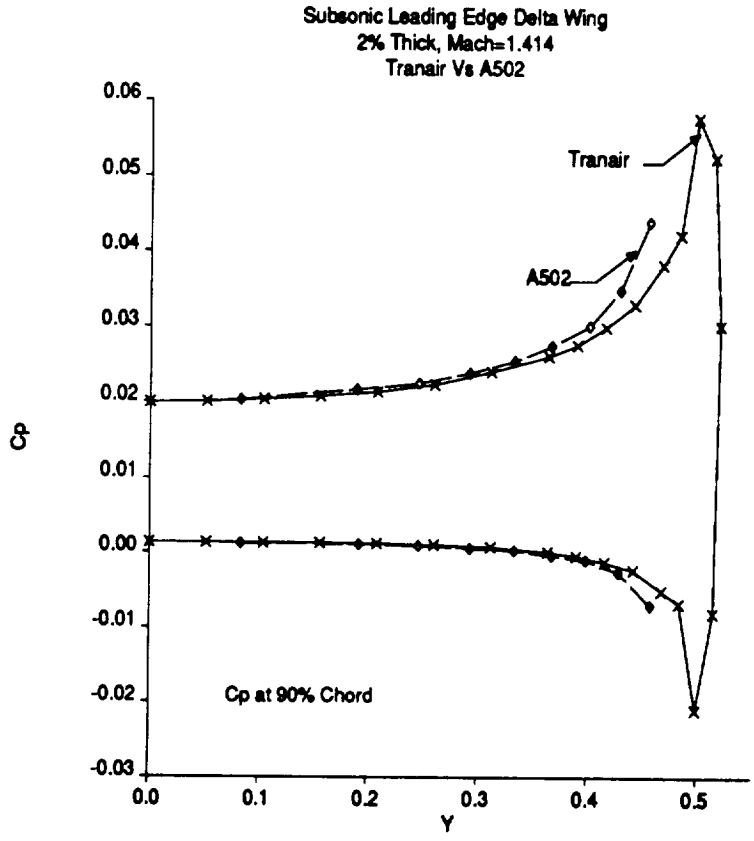


Figure 3.36: Cp Distribution on Subsonic Leading Edge Delta Wing at Mach 1.414. TRANAIR vs A502 Solution.

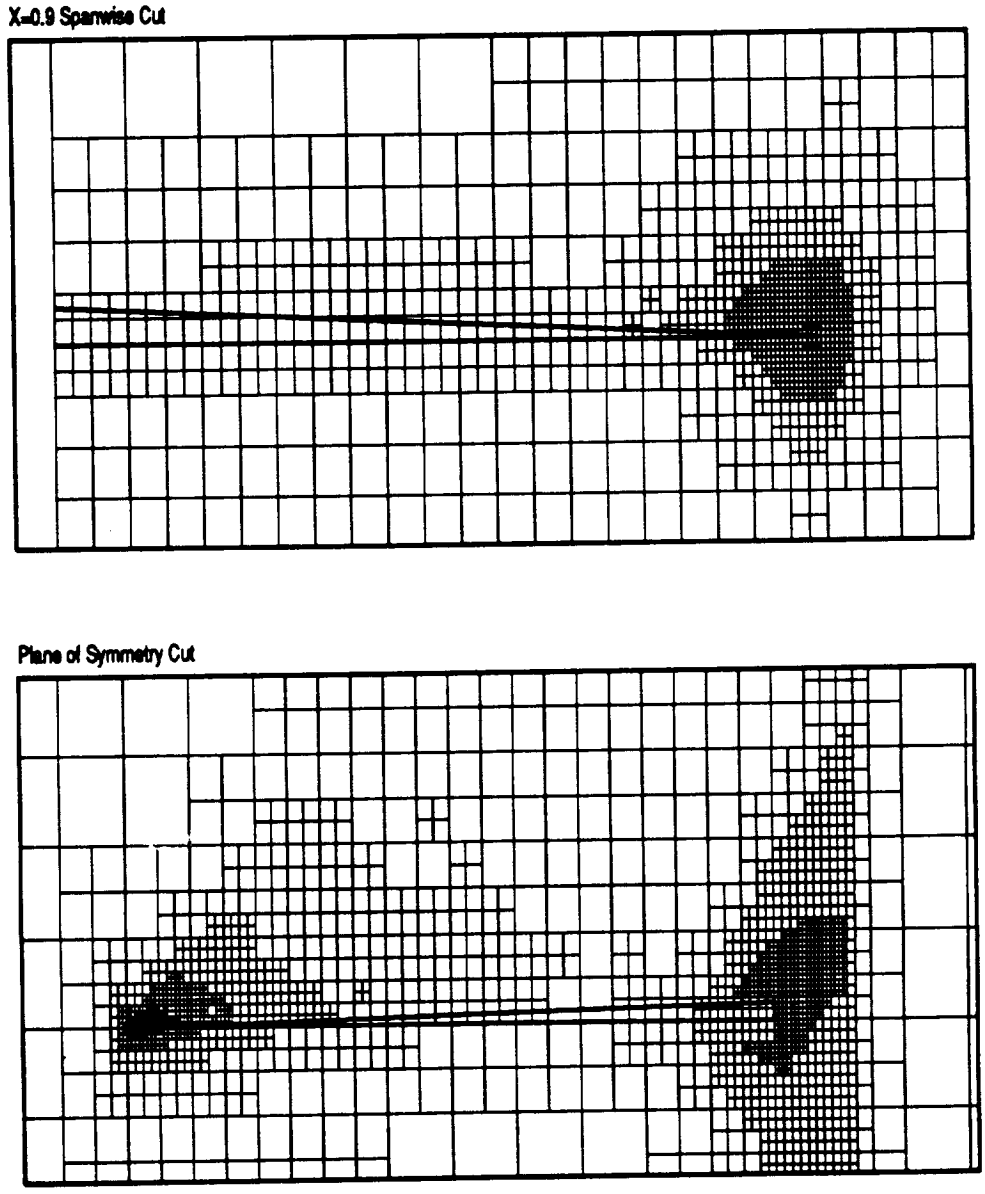


Figure 3.37: Final Computational Grid for Subsonic Leading Edge Delta Wing Configuration.

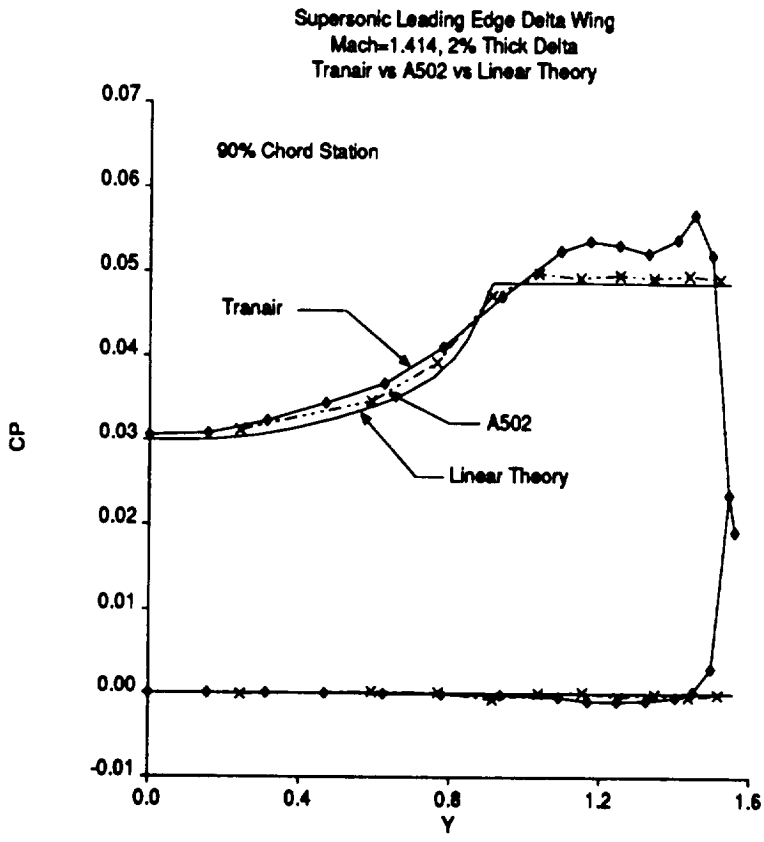


Figure 3.38: Cp Distribution on Supersonic Leading Edge Delta Wing at Mach 1.414. TRANAIR vs A502 Solution.

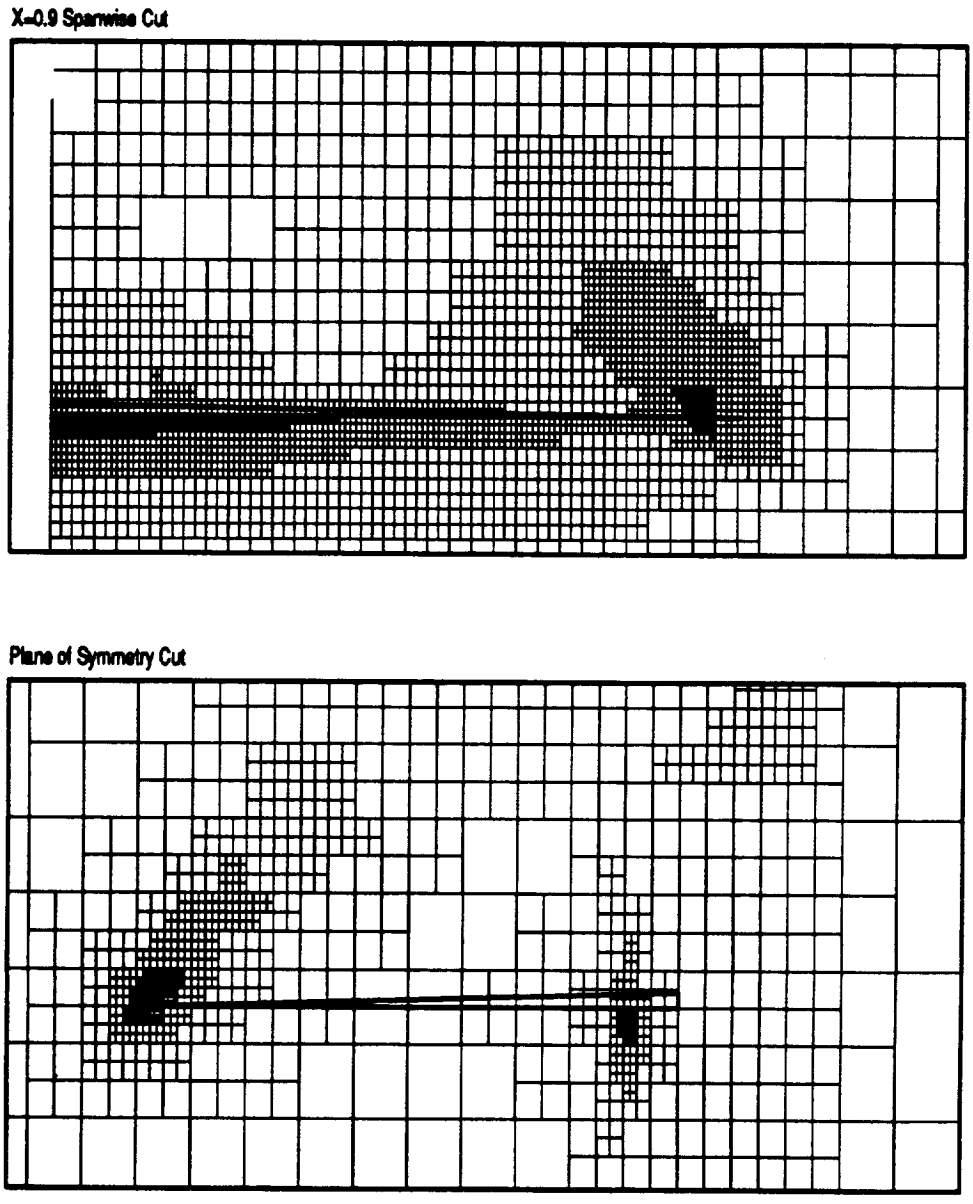


Figure 3.39: Final Computational Grid for Supersonic Leading Edge Delta Wing Configuration.

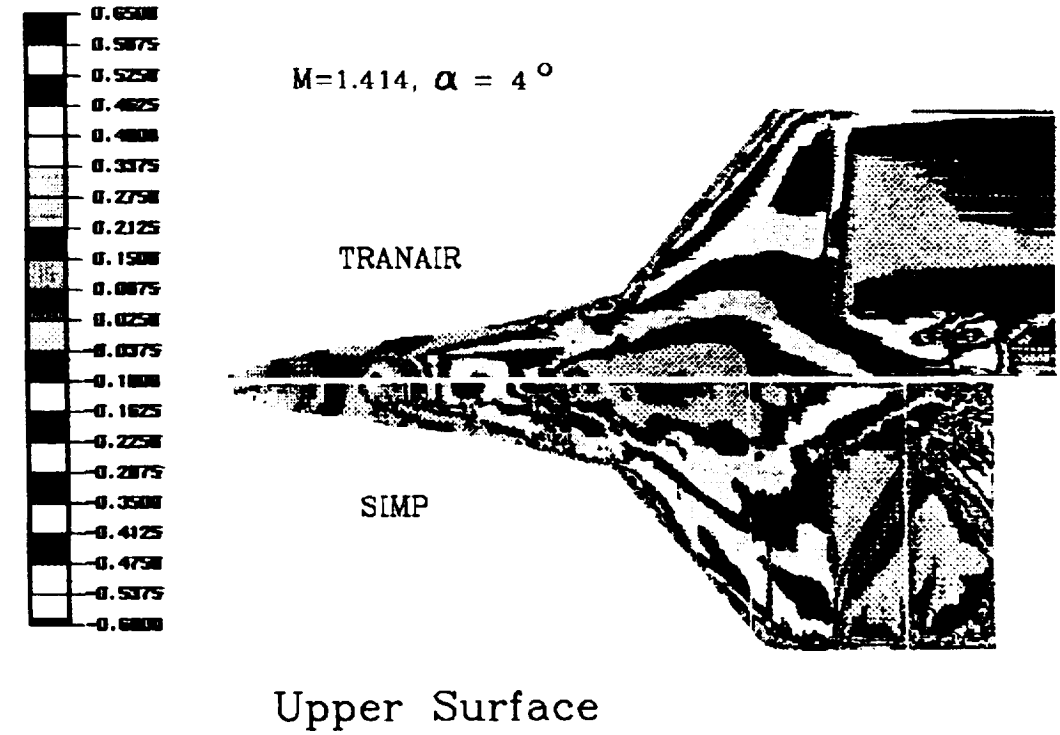


Figure 3.40: Cp Distribution on Upper Surface of F16, $M_\infty = 1.414$, TRANAIR vs SIMP.

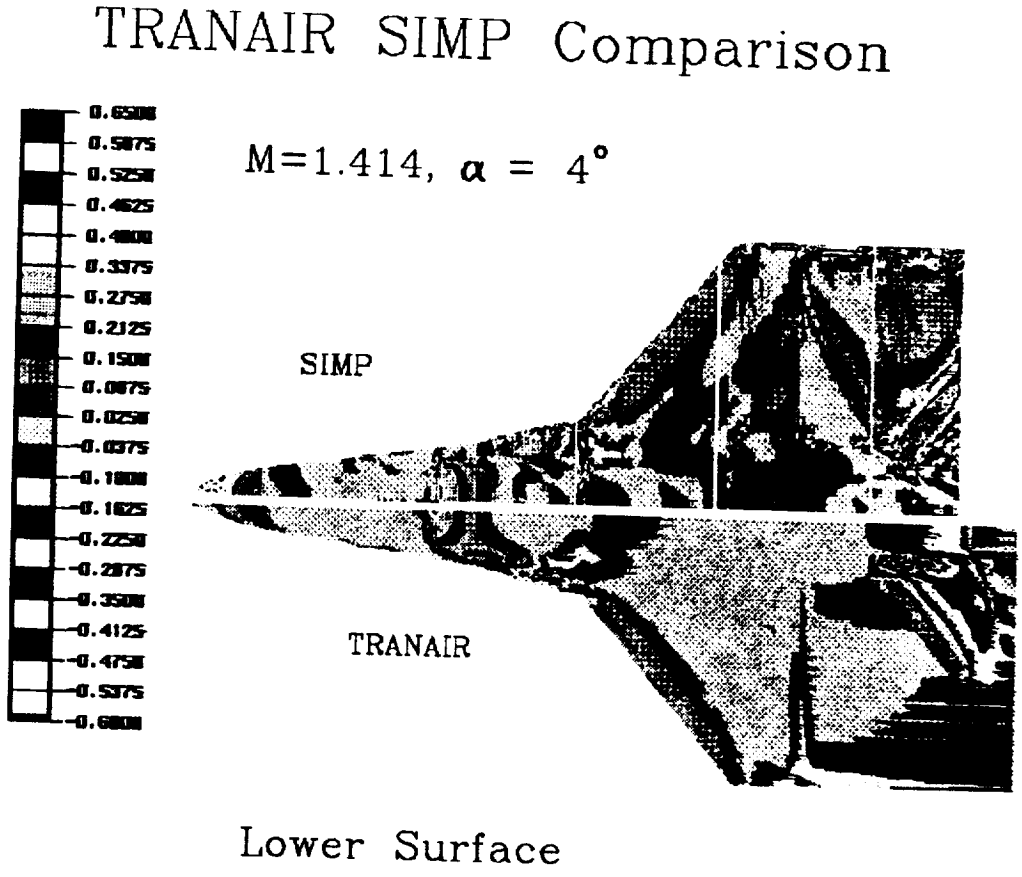


Figure 3.41: Cp Distribution on Lower Surface of F16, $M_\infty = 1.414$, TRANAIR vs SIMP.

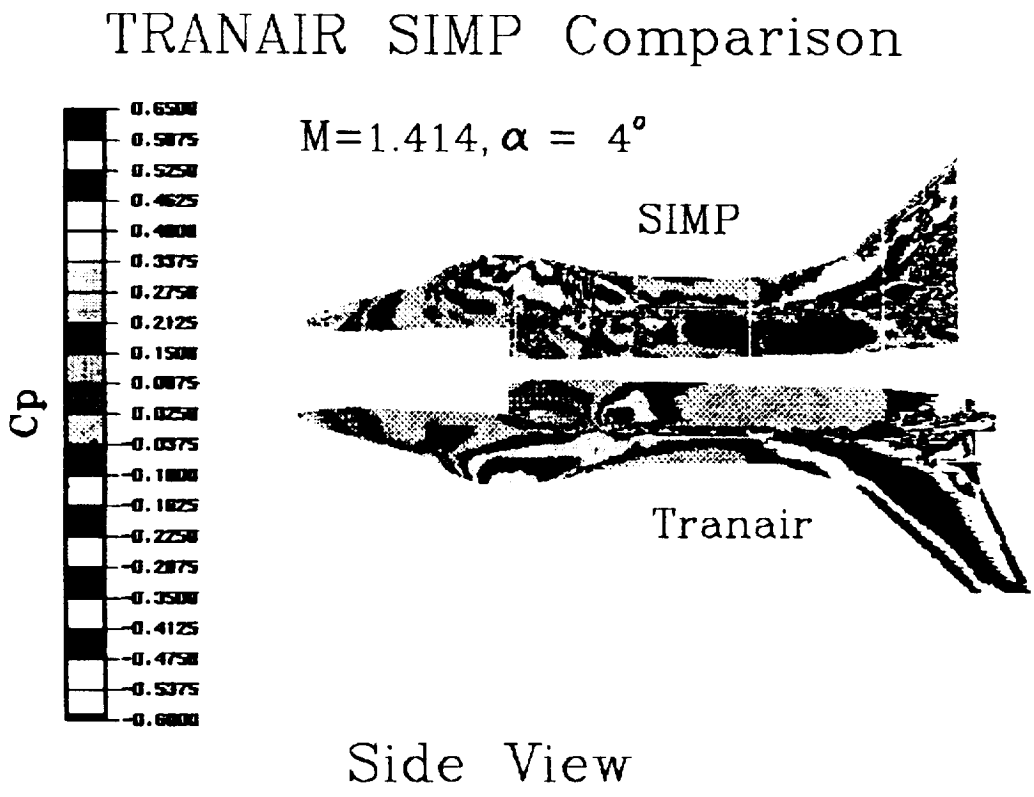


Figure 3.42: Cp Distribution on Side View of F16, $M_\infty = 1.414$, TRANAIR vs SIMP.

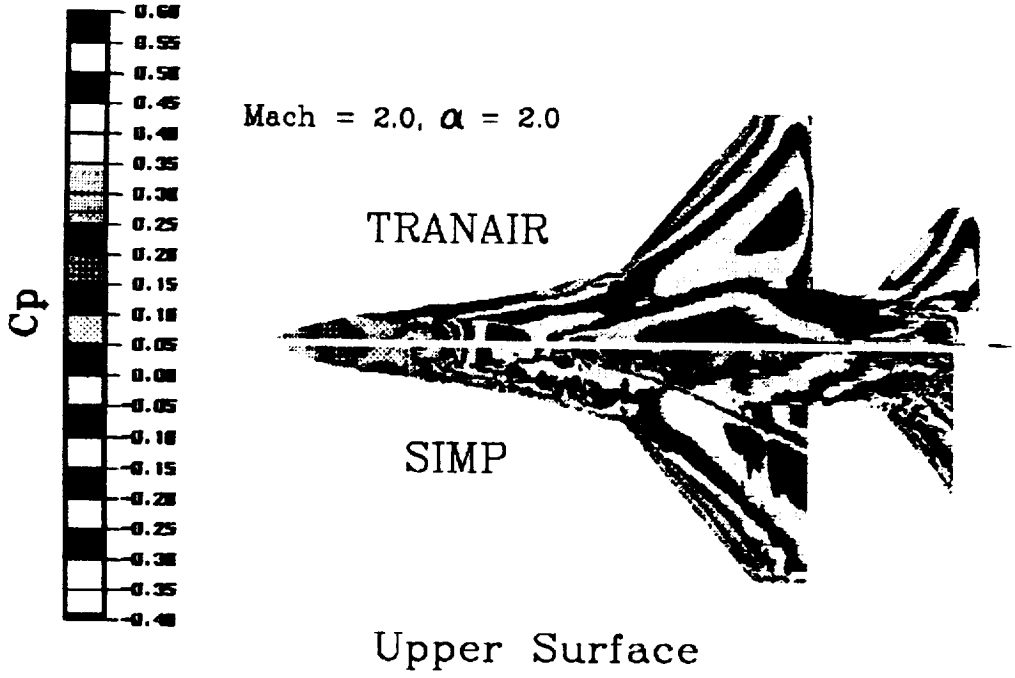


Figure 3.43: Cp Distribution on Upper Surface of F16, $M_\infty = 2.0$, TRANAIR vs SIMP.

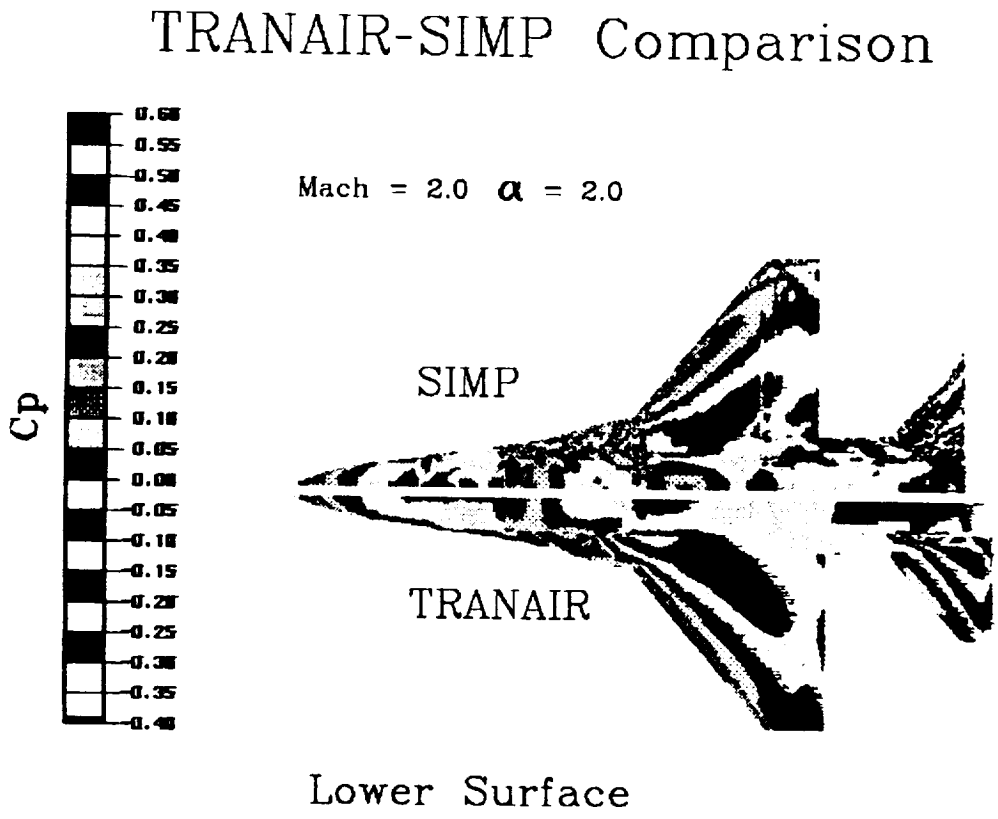


Figure 3.44: Cp Distribution on Lower Surface of F16, $M_\infty = 2.0$, TRANAIR vs SIMP.

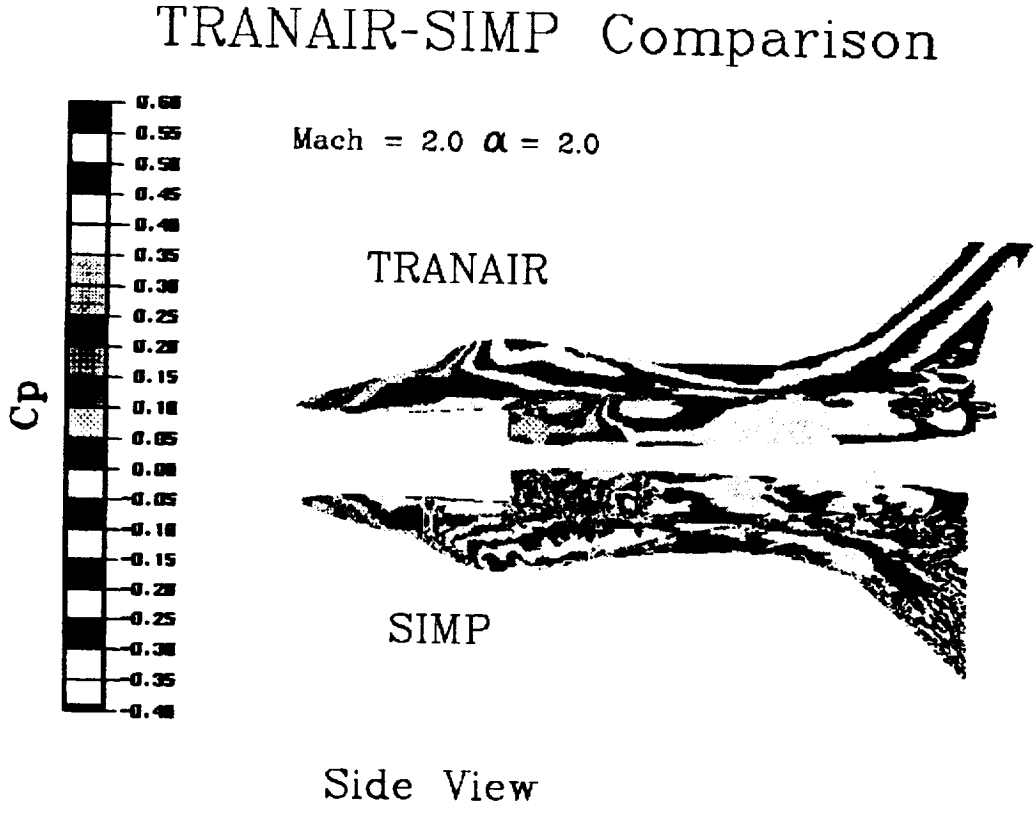


Figure 3.45: Cp Distribution on Side View of F16, $M_\infty = 2.0$, TRANAIR vs SIMP.

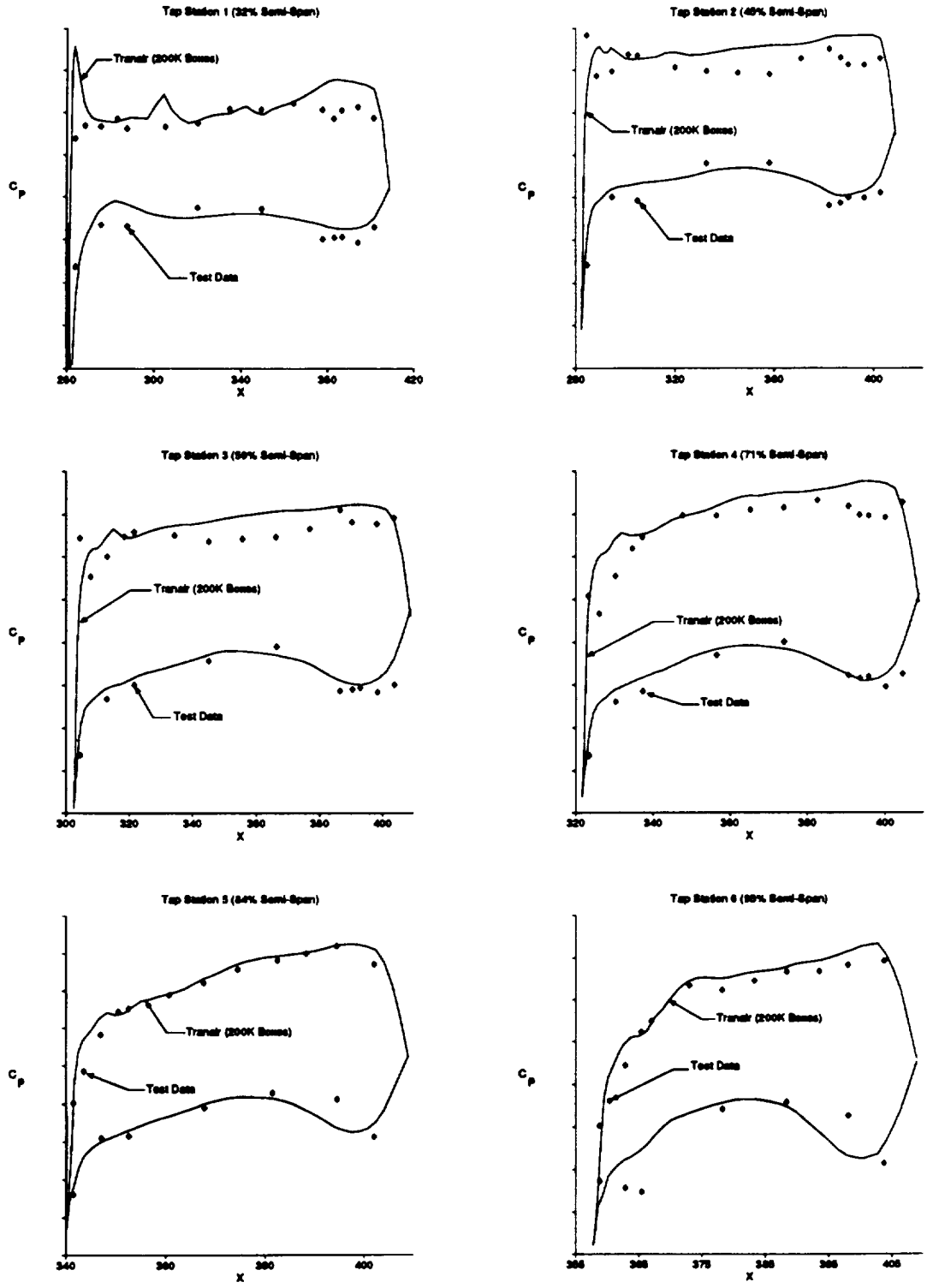


Figure 3.46: Cp Distribution on F16 Configuration with Tip Missiles, $M_\infty = 1.2$ and $\alpha = 4°$, Comparison of TRANAIR with Test Data.

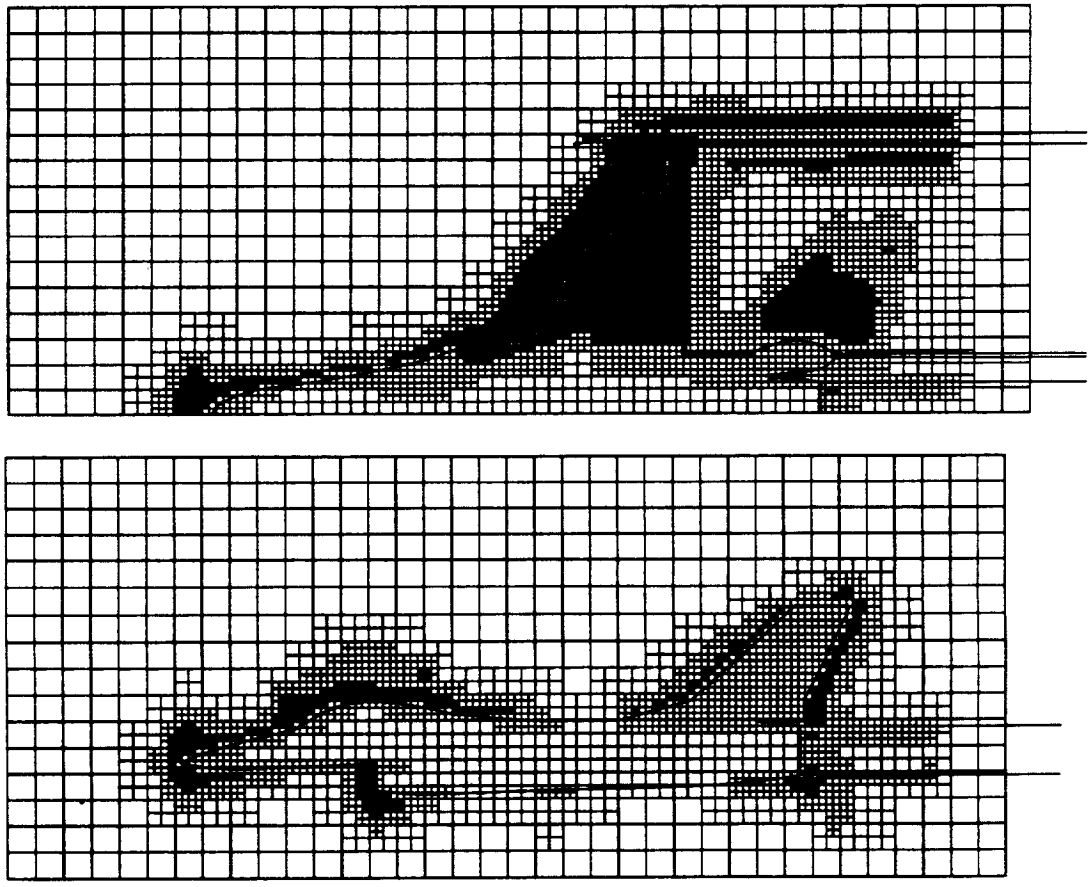


Figure 3.47: Representative Cuts Through Computational Grid for F16 Configuration With Tip Missile.

3.3.4 Bow Shocks

To test the ability of the solution adaptivity in TRANAIR to capture bow shocks, the cone sphere configuration was reversed and the case was re-run under the same free stream conditions. Figures 2.21 through 2.24 illustrate the computational grids generated by TRANAIR for this configuration. The bow shock is captured up to the point that the shock becomes oblique. Five cycles of grid adaptation were used to obtain the solution in Figure 3.48. The early refinements were primarily attracted to the subsonic region between the configuration surface and the bow shock. After the third cycle of solution adaptive refinement, the estimated errors in the subsonic region had been reduced enough so that the error indicators in the bow shock dominate the refinement process. After five cycles of solution adaptive gridding, the fringes of the refinement along the bow shock are the only significant areas indicated for further refinement. Figure 3.48 compares TRANAIR predictions for the location of the bow shock over a range of Mach numbers from 1.01 to 2.8 with experimental data published in some standard fluid flow textbooks [75], [76]. TRANAIR predicts bow shock locations somewhat downstream of the experimental curves, but there is some significant scatter in the experimental data as indicated by the three positions derived from shock position data for 0.25", 0.5" and 1.0" spheres.

To test whether the solution adaptivity could detect and capture a bow shock in a realistic configuration, a TRANAIR analysis was performed on the F16 configuration with underwing fuel tanks present. Five cycles of solution adaptivity were performed. Figure 3.49 illustrates the grid generated by TRANAIR for a chordwise cut through the wing and underwing tank. There is clearly a bubble of subsonic flow both in front of the underwing tank and in front of the strut supporting the tank. The solution adaptive gridding has clearly detected the bow shock and resolved it up to the sonic point, where the bow shock weakens and becomes oblique. This behavior is very similar to that observed for the sphere-cone.

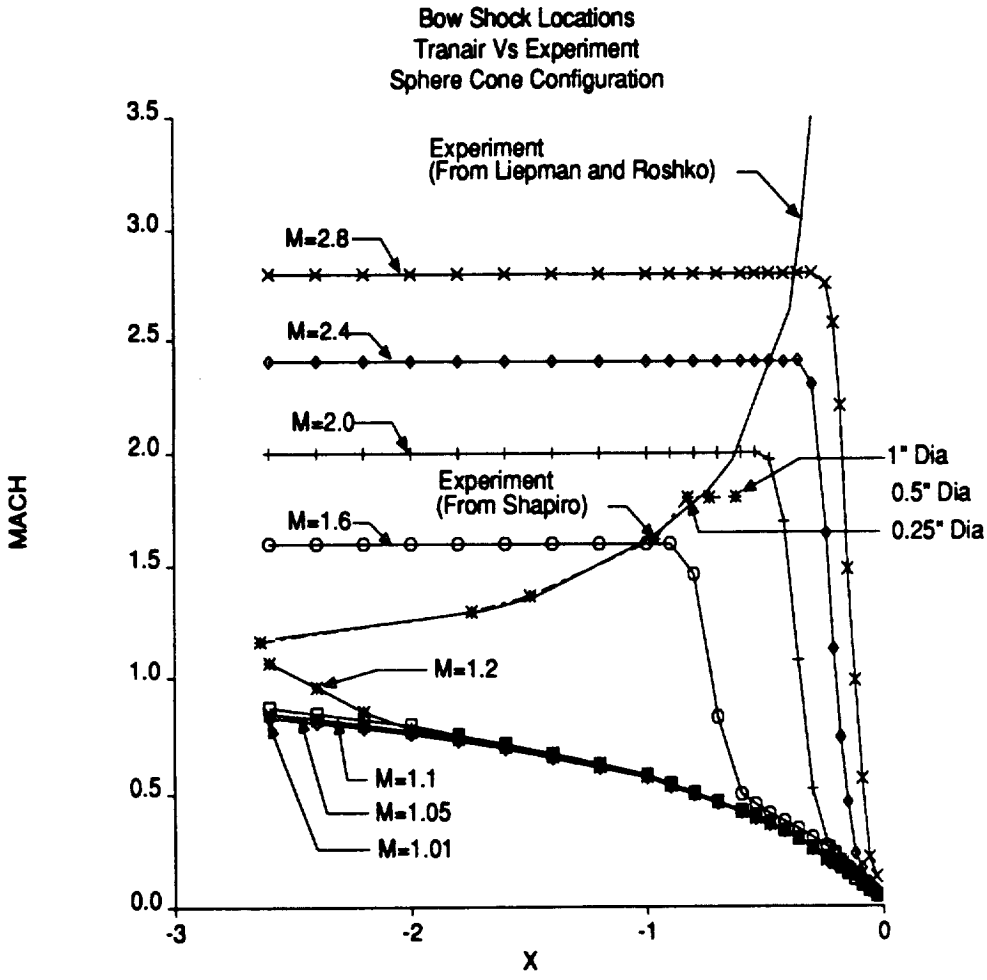


Figure 3.48: Cp Distribution on Sphere-Cone Configuration, $M_\infty = 1.414$.

F16 Tank and Missile - Mach = 1.2

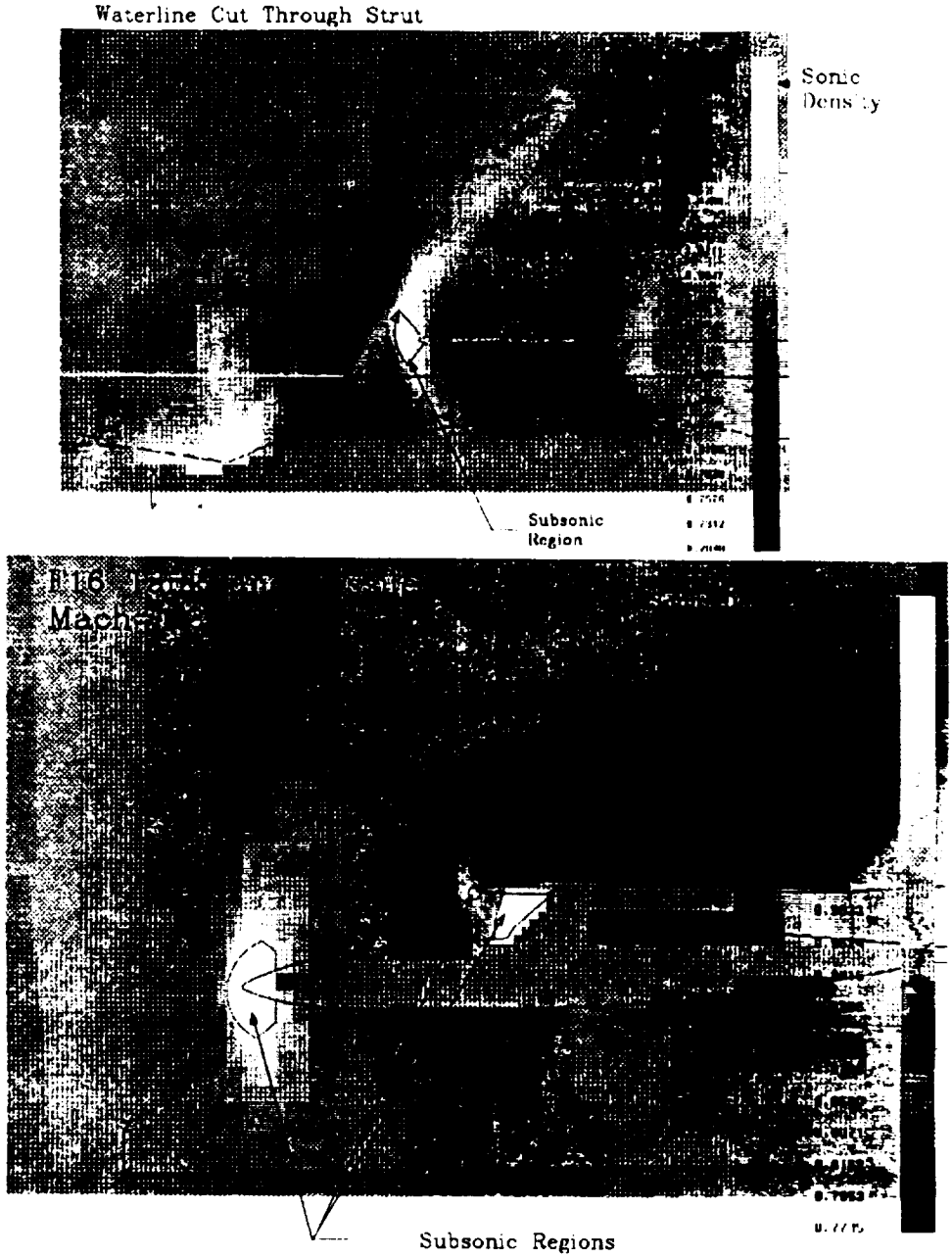


Figure 3.49: Computational Grid for F16 Configuration With Tip Missiles and Wing Tanks, $M_\infty = 1.2$, $\alpha = 4°$.

3.3.5 General Observations.

For supersonic free stream flows, particularly in the case of supersonic leading edges, surface pressure peaks at leading edges tend to be suppressed, compared to subsonic free stream cases or to subsonic leading edges. In addition the surface pressure distributions tend to be flatter and and have smoother gradients. On account of this it is possible to obtain solutions for supersonic free streams using somewhat smaller computational grids than for subsonic free streams. Many of the results obtained in this section used fewer than 100,000 boxes in the entire grid. However, in supersonic free stream flows, aft-facing portions of the configuration can easily generate very large gradients and tend to attract a disproportionate degree of refined grid. In addition, whenever a bow shock is present, the velocity gradients become very large in the region between the bow shock and the configuration surface. This also tends to attract grid refinement in preference over the field discontinuity itself, particularly when the shocks become oblique. It is quite possible to capture bow shocks with the current solution adaptive gridding in TRANAIR, but a larger number of cycles of solution adaptivity or some extra guidance by the user is advisable to make sure that the grid refinement goes into regions of more importance to the application. As an extreme, but very realistic example: left to its own preferences, TRANAIR would refine the cutout regions of the F16, particularly the cutouts near the horizontal tail, in preference to any other portion of the configuration. In the present state of the code, it is recommended that, for supersonic free stream flows, some initial analyses be performed on fairly coarse grids (up to 100,000 boxes, and three to four cycles of solution adaptivity). After these results are obtained, the solution and grid can be examined (for example, by using the TGRAF program, described in Appendix B of the User's manual), and appropriate user directives concerning regions for extra emphasis and de-emphasis can be defined.

Chapter 4

FUTURE DIRECTIONS

The ultimate goal of this work is to offer designers a reliable, general purpose, full configuration flow analysis tool. For this purpose we have preferred to start with a reasonably simple nonlinear physical model (the full potential equation) where we are certain that such an objective can be met. At this time we have implemented an approach to solving the full potential equation into a computer code called TRANAIR. Results obtained from this code on a variety of configurations have shown that TRANAIR is capable of achieving this ultimate goal. In fact TRANAIR is currently being used successfully by many engineers[49], [50], [82] to analyze complex configurations.

However, TRANAIR is by no means a finished product. First, a variety of improvements must be made before TRANAIR can really be thought of as a reliable, general user tool. Second, should these improvements be made, the next order of business would be to enlarge the scope of problems addressed by TRANAIR. Currently TRANAIR allows regions of differing total temperature and pressure, which is important for simulating propulsion effects. An additional capability which would be desirable is the capturing of vortex, or wake sheets. In theory this can be done within the framework of a full potential approach. (However, it can be argued that a method which allows differing total pressures and temperatures as well as wake capturing is three-fourths of the way to an Euler method.) Another highly desirable capability would be the simulation of boundary layer effects. Such effects are extremely important in the transonic flow regime. A final capability, which could greatly aid the design process is an optimization algorithm, allowing the user to specify certain desirable flow features which TRANAIR would then try to achieve with geometry adjustments.

In this chapter, we discuss some ideas on future efforts to improve TRANAIR. In Section 4.1 we discuss efforts required to improve the reliability, accuracy and efficiency of the current code. In the remaining sections we discuss improvements in capability. In Section 4.2 we explore some of the issues involved in extending TRANAIR to solve the full Euler equations, which would automatically yield a wake capturing capability. An alternate, and preferred approach, still within the framework of the full potential equation is described in Section 4.3. In Section 4.4 we briefly discuss the addition of a boundary layer capability to TRANAIR. In Section 4.5 we discuss implementation of design and optimization capability.

4.1 IMPROVEMENTS TO THE METHOD

4.1.1 Reliability and Efficiency Improvements

In the current implementation of the GMRES algorithm the iterative procedure continues until the preconditioned residuals are reduced by a fixed fraction (assuming the maximum allowable number of iterations is sufficiently high). Because of the use of a drop tolerance in the sparse solver the condition number of the preconditioned linear system solved by GMRES may vary considerably. Such a condition number should be taken into account when assigning the above fraction. There are various ways for estimating the condition number. For example, comparing the reduction in real residuals with the reduction in preconditioned residuals can give a lower bound.

On occasion Newton's method has trouble converging on coarse grids, but does quite well on fine grids. This usually happens when the configuration involves small diameter plumes of differing total pressure and temperature. We suspect that on coarse grids the plume geometry is not well resolved, leading to a nearly singular boundary value problem. It would be fairly simple to ensure that the coarsest grid always resolves small scale geometrical features. Moreover, the coarse grid should be the minimal grid which does this, as denser grids may fix the shocks in the wrong locations.

In the case of supersonic free stream flow a local free jet boundary condition (i.e. zero perturbation potential) is currently imposed on the top, bottom and side faces of the computational box. Such a condition is certainly better than a solid wall condition, but both cause wave reflections back into the computational box. No local boundary condition is entirely accurate, but an outgoing wave condition is undoubtedly superior to a free jet condition and should be implemented.

Currently the drop tolerance used by the sparse solver is assigned by the user based on the knowledge that a drop tolerance somewhere between .001 and .0001 has generally worked in the past. However, even in this range fill-in (and hence SSD usage) can vary considerably. This is not an issue that the user should have to worry about. The user should only specify the size of the SSD storage available, and the code should then adaptively determine a drop tolerance which would lead to a decomposition of roughly this size.

Currently the off diagonal terms in the sparse matrix decomposition are dropped when their magnitude is smaller than the drop tolerance times the magnitude of the corresponding column (or row) diagonal. This works reasonably well, but many improvements which would substantially reduce decomposition costs and storage are possible. For example, the terms which are dropped could be added to the diagonal or else the diagonal could be augmented by a fixed factor to improve stability when poor conditioning is suspected.

Computation of the finite element and upwinding operators comprises from one-third to one-half of the run cost of the current code. A good share of this cost could be eliminated if operators associated with T-boxes which do not get refined or derefined could be saved from one grid to the next.

One of the most time consuming aspects of a TRANAIR analysis is the construc-

tion of networks of panels from available configuration lofts. In the future it would probably be best for TRANAIR to get its surface definition from the lofts directly. Given a loft for a surface patch TRANAIR could interrogate the loft in an adaptive manner to build an unstructured surface triangularization whose density is determined by the estimated local curvature. Because the cost of solution depends very weakly on surface discretization it would be possible to make the initial discretization sufficiently dense to accommodate the finest grid. However, the unstructured nature of the surface discretization would require the user to supply additional information concerning the nature of output of surface flow quantities.

The sparse solver is used as a preconditioner for unknowns in the reduced set. Currently the reduced set contains all unknowns except for those located at subsonic global grid nodes. These unknowns are preconditioned by the much cheaper Poisson solver. Using standard elliptic multi-grid methods it should be possible to extend the Poisson solver to handle refined grids. This would allow the elimination of subsonic refined grid unknowns from the reduced set as long as they were not located at the boundary. In many instances the size of the reduced set would decrease by as much as 50% leading to a substantial reduction in the costs associated with the sparse solver.

The process of interpolating from a coarse grid to a fine grid has already been developed to facilitate grid sequencing. Such a process could also be the basis for implementing a multigrid solution procedure. We recommend a limited implementation wherein the sparse solver preconditioner calculated on the next to finest grid is also used on the finest grid. This can be done by collecting residuals onto the coarser grid, using the sparse solver preconditioner as a smoother on this grid, distributing the corrections to the fine grid and locally smoothing the fine grid residuals. Since the fine grid is generally at least twice as large as the next coarsest grid the CPU and storage savings of such a procedure could be very significant.

4.1.2 Upwinding Improvements

Currently, the "entropy condition" for ensuring compression shocks is achieved through a first order density or mass flux retardation procedure. This procedure occasionally causes the reliability problems and affects efficiency and accuracy of the code. The first problem is that the upwinding is only first order whereas the remainder of the method is second order. This forces the grid to be finer in supersonic zones than in subsonic zones, although in supersonic regions the adaptive feature of the code will limit grid refinement to regions with high flow gradients, somewhat alleviating the problem. One course of action would be to implement a second order accurate upwinding algorithm. Historically such algorithms have not been very robust, especially for complex geometries. A better course of action might be to develop a first order algorithm with a smaller error coefficient. In fact, flux biasing (or retardation) seems to be much superior to density retardation in this regard and there appear to be possibilities of improving it by taking variations of stream tube areas into account. Unfortunately, flux biasing can be shown to be singular when the flow becomes one dimensional and the Mach number oscillates about Mach 1. This makes flux biasing substantially less robust than density biasing in practice and it will first be necessary

to modify the biasing formula in a manner which simulates the use of a subsonic cutoff Mach number in density retardation formulas.

The potential flow assumption produces errors in normal shock strength when the Mach number ahead of the shock is greater than approximately 1.4. Hafez[77] has developed a correction to account for entropy production, which apparently allows more realistic flow simulations at higher Mach numbers. This correction should be implemented and tested in TRANAIR.

During the Newton iteration process, velocities emanating from boundary surfaces may appear. If these velocities correspond to supersonic Mach numbers then upwinding cannot be performed properly and the problem temporarily becomes singular. This leads to a breakdown in the solution procedure. In the case of an engine exit, where the velocity is supposed to emanate from the exit surface, we upwind densities next to the exit to a fictitious density which corresponds to free stream Mach number. This works quite well if the exit Mach number should be subsonic and the free stream Mach number is also subsonic. The linearized problem is then non-singular and within several Newton steps the exit Mach number recovers to a subsonic value. If the exit Mach number is supposed to be supersonic, then the user must specify this Mach number (although not all values are feasible). We have not yet developed the input formats to allow the user to do this, but we have demonstrated that by retarding the density to a fictitious density corresponding to this specified Mach number, the exit velocity eventually settles to the proper value. Therefore the input formats should be developed.

We have not yet developed a strategy when the surface is a solid surface or a wake rather than an exit. Density is continuous across wakes so the problem may be handled by upwinding to densities on the opposite side. In the case of solid walls where the local flow should be supersonic it would be difficult to employ a fictitious density, since the true local Mach number is only known upon solution. It might be better to stop the flow direction anomaly from arising altogether by damping the Newton method based on velocity direction changes.

4.1.3 Solution Adaptive Grid Improvements

The ultimate goal of solution adaptive grid refinement is to produce an accurate flow simulation at a low computational cost with minimal user intervention. In its present form TRANAIR makes significant progress towards this goal. In working with the solution adaptive process a number of ideas have emerged which will carry TRANAIR considerably further towards this goal. Because these areas have not been thoroughly investigated to date, the ideas discussed below have not yet been implemented in the code. It is expected that after further analysis and careful testing, their implementation will produce significant improvements in the effectiveness of solution adaptive gridding (with minimal user intervention).

In the first place, TRANAIR would provide more effective solution adaptive grids if it did a better job of exploiting the physical aspects of the flow. For example, if in some region of the flow the local Mach number exceeds the fictitious gas Mach number, the full potential equation is a poor approximation to the flow. Although

the estimated errors in the neighborhood of such regions may be quite high, creating higher grid resolution in such areas provides little benefit in terms of obtaining a more meaningful engineering answer to the flow problem. Thus refinement should be limited to regions where the flow remains physically sensible. In addition, where real discontinuities in the flow field occur (i.e., at a shock), no matter how much grid refinement is applied in the vicinity of the shock, jumps in the velocities will still remain, and the presently implemented error indicators will remain large. Thus some form of automatic local "shock limiting" is desirable to prevent grid refinement beyond what makes sense for a given engineering application. At present there is such a limiter but it is determined from user input. Shocks (particularly normal shocks) can easily be recognized by the code, and thus it would be fairly straightforward to automatically introduce limits to grid refinement wherever shocks occur, without any user specification.

Another easily recognized physical phenomenon related to shocks is whether the flow in a region is undergoing expansion or compression. With the current implementation of solution adaptivity, grid refinement tends to be attracted to expansion regions (like the leading edges of a wing). Ultimately, this is a desirable phenomenon, but when excessive grid is attracted to an expansion region, it can happen that latent features of the flow field (for example, oblique shocks) do not attract adequate grid density. The result can be a smooth pressure distribution which hides the existence of the phenomenon. (The ONERA M6 wing has proven to be a very good case for analyzing this difficulty). Thus it appears to be desirable to suppress grid refinements in expansion regions in the earlier steps of solution adaptive gridding. This encourages grid refinement in regions where these latent features might occur. Only in the final stages of solution adaptive gridding should the grid be permitted to cluster in expansion regions. The present code allows for suppression of grid refinement in expansion regions, but such regions must be identified *a priori* by a user.

Some exploration has been made of alternative error indicators. The present implementation bases these on discontinuities in velocity magnitudes from cell to cell. This indicator provides high error estimates when the velocity magnitude jumps, but such error estimates are relatively small when a jump is due to a turning of the velocity (as occurs for oblique shocks). It is possible that an error indicator based on the change in the direction of the velocity would provide earlier emphasis of more latent flow features, such as oblique shocks.

In the early applications of solution adaptivity, the same type of error estimates and the same grid refinement strategy have been applied for subsonic, transonic and supersonic free stream flows. It is quite likely that because of the physical differences in these problems, different solution adaptive strategies may be beneficial.

Ultimately an overall strategy for solution adaptive refinement will emerge from these ideas which will consist of a number of error indicators along with an appropriate weighting as the solution adaptive gridding continues in order to:

- capture the latent features of the flow early, when the computational costs are low;
- recognize regions where true discontinuities occur and limit refinement to length

scales that make good engineering sense;

- provide sufficient grid resolution in expansion and stagnation areas to produce accurate solutions on the final grid;
- automatically recognize regions of non-physical flow features and avoid wasting grid resolution there;
- automatically (or with very little user intervention) detect regions where there is little interest in resolving high flow gradients such as wing tips, but at the same time, allowing the user to study these areas if that is the design goal;
- provide a near optimal strategy for solution adaptive gridding for subsonic, transonic, and supersonic flows using only knowledge of the free stream Mach number.

The elements of this final strategy have been identified and some issues have been explored. Significant improvements can be made in the adaptive gridding, resulting in more efficient and accurate solutions.

4.1.4 Higher Order Elements

Higher order finite elements are currently being developed for structures applications [78],[79] with remarkable success. Incompressible Navier-Stokes calculations are also being attacked with these methods [80],[81]. For smooth problems, these methods offer exponential order convergence (better than any algebraic order of convergence) in the number of unknowns. For problems with singularities, they offer substantially better algebraic rates of convergence. These methods have the same rate of convergence as spectral methods, but are not subject to the same limitations such as rectangular domains and separable grids.

Thus, there is the potential for large savings in CPU time and storage in TRANAIR with the proper use of higher order finite elements. In particular, in solving the full potential equation the important flow quantities are defined in terms of the velocity, which is only first order accurate in the mesh spacing locally. Since the velocity is first order, to achieve a factor of two reduction in error currently requires eight times as many elements. With second order velocity only three times as many elements are required. Thus, the payoff of higher order methods is great.

Higher order basis functions that have successfully been used include Tchebychev polynomials, Legendre polynomials, and other higher order polynomials. Continuous basis functions seem to be much more flexible than ones with more degrees of inter element continuity. In TRANAIR, one would implement a triquadratic element basis function for the potential. This would require a trilinear approximation to the density. The density would be an interpolating polynomial fitting four points in every element. Integrals would probably best be evaluated with numerical quadrature rules. A sophisticated adaptive strategy would be needed to determine where to use these higher order elements and where to use the currently implemented trilinear elements.

An unresolved issue is how to do higher order upwinding in supersonic regions. We can observe however, that a second order upwinding of density could be achieved without any more element connectivity information than is currently in TRANAIR.

There have recently been significant advances in iterative methods for the moderately sparse linear systems resulting from these discretizations. The solution technique can use a recent result of Babuska that the lowest order stiffness matrix is an excellent preconditioner for the higher order finite element problem. Thus, the current Jacobian matrix calculation and decomposition could be used as the direct solver preconditioner. If successful, this would save significant coding and computational expenses.

Thus, the broad outlines of a higher order method for TRANAIR have been thought out. Such a method could provide much greater accuracy at reasonable cost than current methods which are all second order in the potential. This could enable the accurate calculation of such sensitive measures of performance as inviscid drag, which current methods can not predict.

4.2 EULER FORMULATION

4.2.1 Properties of Euler Equations

The steady state Euler equations express conservation of mass, momentum and energy as follows:

Conserved Quantity	D. E.		FLUX			
mass	$\vec{\nabla}\cdot\vec{W}$	$=0$	$\vec{W}$	$=$	$\rho\vec{U}$	(4.1)
momentum	$\vec{\nabla}\cdot m$	$=0$	m	$=$	$\vec{W}\vec{U}^{\,T}+pI$	(4.2)
energy	$\vec{\nabla}\cdot\vec{E}$	$=0$	$\vec{E}$	$=$	$H\vec{W}$	(4.3)

In the first column we display the quantity conserved across discontinuity surfaces, i.e. the normal component of flux, in the second, we display the differential equation for each conservation law, and in the third, we display the relevant flux. To complete the description we define total enthalpy, H, and entropy, S.

$$\textit{enthalpy definition}\quad H = \frac{\gamma p}{(\gamma-1)\rho}+\frac{1}{2}q^2\ ,\ q=\left|\vec{U}\right| \tag{4.4}$$

$$\textit{entropy definition}\quad \frac{p}{p_\infty} = e^{(\gamma-1)S}\left(\frac{\rho}{\rho_\infty}\right)^\gamma \tag{4.5}$$

In order to eliminate the possibility of expansion shocks we need an entropy condition which we choose to introduce via artificial pressure, i.e. we redefine m as

$$m=\vec{W}\vec{U}^{\,T}+\tilde{p}I\,\vec{\epsilon}\cdot\vec{\nabla}_- p \tag{4.6}$$

Here $\vec{\nabla}_- p$ is a backward derivative and $\vec{\epsilon}$ a vector in the direction of $\vec{U}$ having magnitude on the order of the grid size. The $\vec{\epsilon}$ is non-zero only in supercritical regions. For the discussion which follows it suffices to ignore artificial pressure.

To better understand the Euler equations we can recombine Eqns. (4.1)-(4.3) in the following ways

$$(energy) - H(mass) \equiv \vec{W} \cdot \vec{\nabla} H = 0 \tag{4.7}$$

$$(q^2 - H)mass - \vec{U} \cdot H(momentum) + (energy) \equiv \frac{p}{\rho} \vec{W} \cdot \vec{\nabla} S = 0 \tag{4.8}$$

$$(momentum) - \vec{U}\,(mass) \equiv - \vec{W} \otimes \vec{\omega} - p \vec{\nabla} S + \rho \vec{\nabla} H = 0 \tag{4.9}$$

Here $\vec{\omega}=\vec{\nabla} \otimes \vec{U}$ is the vorticity vector. Equation (4.9) can be rewritten in a better way by introducing the concept of swirl, i.e.

$$G \equiv \frac{\vec{W} \cdot \vec{\omega}}{\rho^2 q^2} \equiv swirl \tag{4.10}$$

Then Eqn. (4.9) becomes

$$\vec{\omega} = G \vec{W} + \frac{p}{\rho^2 q^2} \vec{W} \otimes \vec{\nabla} S - \frac{1}{\rho q^2} \vec{W} \otimes \vec{\nabla} H \tag{4.11}$$

Using Eqn. (4.1) and the fact that $\vec{\nabla} \cdot \vec{\omega} = 0$ we obtain an equation for G by taking the divergence of Eqn. (4.11), i.e.

$$\vec{W} \cdot \vec{\nabla} G = - \vec{\nabla} \cdot \left[\frac{p}{\rho^2 q^2} \vec{W} \otimes \vec{\nabla} S - \frac{1}{\rho q^2} \vec{W} \otimes \vec{\nabla} H \right] \tag{4.12}$$

Now let us assume we have an initial estimate of $\vec{W}$, and let $\vec{U}$, S and H be the fundamental unknowns. Equation (4.7) is a convection equation for H and states that H is constant along streamlines. If H is specified at the head of every streamline then H may be found at every point in the flow field. In particular if H is the same constant at the head of every streamline then H will be identically constant in the flow field and the flow will be isoenergetic. Mechanisms which produce non-constant H include propellers and jet engines. The appropriate value of H must be specified at the head of each streamline leaving these mechanisms. From Eqn. (4.8) we see that S also satisfies a convection equation. Entropy must also be specified at the exit of propulsion devices. However, entropy also has field sources in the case that dissipation is present, e.g., when Eqn. (4.6) is operable. Then Eqn. (4.8) will have a non-zero right hand side. These "convection" sources should be negligible except at a shock. Once H and S are known, Eqn. (4.12) becomes a convection equation for G with a specified right hand side field source. This equation can be integrated to give G everywhere in the flow field once G has been specified at the head of every streamline. If S and H are constant in the flow field, then the only source of swirl is via boundary conditions at the head of streamlines. Again, propulsion devices produce swirl, but another major source is the Kutta condition at trailing edges. From Eqn. (4.11) we

see that in the absence of variations in S and H vorticity can only be produced by swirl. In fact, free vortex sheets in potential flow are produced entirely by swirl.

Once S, H and G are found everywhere in the flow field, $\vec{\omega}$ may be determined from Eqn. (4.11). If we decompose $\vec{U}$ into a scalar and vector potential

$$\vec{U}=\vec{\nabla}\,\Phi+\vec{\nabla}\otimes\vec{A} \tag{4.13}$$

then $\vec{A}$ may be determined by taking the curl of both sides, i.e.

$$\vec{\nabla}^2\vec{A}=\vec{\omega} \tag{4.14}$$

As in potential flow, Φ may now be determined from the mass conservation Eqn. (4.1) and the specified boundary condition on $\vec{U}$. Unfortunately, this is not the same Φ as in potential flow. Even in portions of the field where $\vec{\omega}$ is zero, $\vec{A}$ will not necessarily be zero, since Eqn. (4.14) spreads $\vec{A}$ to the whole flow field.

An approach which casts the Euler equations as a more direct generalization of potential flow is based on the Bateman variational principle. Here we seek a stationary value of a payoff, J, defined by

$$J=-\int\int\int_V p^*dV \quad , \quad p^*=p^*(\vec{U},H,S) \tag{4.15}$$

here p^* is an arbitrary function of $\vec{U}$, H and S.

Let us define

$$\vec{W}=-\frac{\partial p^*}{\partial U} \quad , \quad p=-\frac{\partial p^*}{\partial S} \quad , \quad \rho=\frac{\partial p^*}{\partial H} \tag{4.16}$$

If we choose p^* to be pressure as the usual function of $\vec{U}$, H and S, then Eqn. (4.16) is consistent with the usual definitions of $\vec{W}$, p and ρ. (One could also choose p^* to be the second-order expansion of p about p_∞ , in which case $\vec{W}$ becomes the usual linear mass flux vector used in panel methods. Rolled up vortex sheets are possible with such an approximation, but not shocks.) Taking a variation of J and neglecting higher order terms we obtain

$$\delta J=-\int\int\int_V\left[\vec{W}\cdot\delta\,\vec{U}+p\delta S-\rho\delta H\right]dV \tag{4.17}$$

Let us now use a Clebsch decomposition of the velocity vector, i.e.

$$\vec{U}=\vec{\nabla}\,\Phi+\vec{Q} \quad , \quad \vec{Q}=\mu\,\vec{\nabla}\,\lambda-S\,\vec{\nabla}\,\eta+(H-H_\infty)\,\vec{\nabla}\,\zeta \tag{4.18}$$

Then

$$\begin{aligned}\delta\,\vec{U}=\vec{\nabla}\,\delta\Phi+\;&\delta\mu\;\;\vec{\nabla}\,\lambda+\mu\,\vec{\nabla}\,\delta\lambda\\ -\;&\delta S\;\;\vec{\nabla}\,\eta-S\,\vec{\nabla}\,\delta\eta\\ +\;&\delta H\;\;\vec{\nabla}\,\zeta+(H-H_\infty)\,\vec{\nabla}\,\delta\zeta\end{aligned} \tag{4.19}$$

Hence

$$\begin{aligned}\delta J = \int\int\int_V \; [\;& (\vec{W}\cdot\vec{\nabla}\,\lambda)\delta\mu - (\vec{W}\cdot\vec{\nabla}\,\eta)\delta S \\ &+ (\vec{W}\cdot\vec{\nabla}\,\zeta)\delta H + p\delta S - \rho\delta H]dV \\ +\int\int\int_V \; [\;& (\vec{W}\cdot\vec{\nabla}\,\delta\Phi) + \mu(\vec{W}\cdot\vec{\nabla}\,\delta\lambda) \\ &- S(\vec{W}\cdot\vec{\nabla}\,\delta\eta) + (H - H_\infty)\,\vec{W}\cdot\vec{\nabla}\,\delta\zeta]dV \end{aligned} \tag{4.20}$$

Integrating the second integral by parts,

$$\begin{aligned}\delta J = \int\int\int_V \; [\;& (\vec{\nabla}\cdot\vec{W})\delta\Phi + (\vec{W}\cdot\vec{\nabla}\,\lambda)\delta\mu + (\vec{\nabla}\cdot\mu\,\vec{W})\delta\lambda \\ &+ (p - \vec{W}\cdot\vec{\nabla}\,\eta)\delta S - (\rho - \vec{W}\cdot\vec{\nabla}\,\zeta)\delta H \\ &- (\vec{\nabla}\cdot S\,\vec{W})\delta\eta + (\vec{\nabla}\cdot H\,\vec{W})\delta\zeta]dV \\ +\int\int_S \; [\;& (\vec{W}\cdot\hat{n})\delta\Phi + (\mu\,\vec{W}\cdot\hat{n})\delta\lambda \\ &- (S\,\vec{W}\cdot\hat{n})\delta\eta + (H\,\vec{W}\cdot\hat{n})\delta\zeta]dS \end{aligned} \tag{4.21}$$

Assuming that δJ is stationary we obtain the following equations

$$\vec{\nabla}\cdot\vec{W} = 0 \tag{4.22}$$

$$\vec{W}\cdot\vec{\nabla}\,\lambda = 0 \tag{4.23}$$

$$\vec{\nabla}\cdot(\mu\,\vec{W}) = \vec{W}\cdot\vec{\nabla}\,\mu = 0 \tag{4.24}$$

$$p - \vec{W}\cdot\vec{\nabla}\,\eta = 0 \tag{4.25}$$

$$\rho - \vec{W}\cdot\vec{\nabla}\,\zeta = 0 \tag{4.26}$$

$$\vec{W}\cdot\vec{\nabla}\,S = 0 \tag{4.27}$$

$$\vec{\nabla}\cdot(H\,\vec{W}) = 0 \tag{4.28}$$

The first equation is the mass equation and the last is the energy equation. The others are then equivalent to the momentum equation as can be seen by substituting Eqn. (4.18) into Eqn. (4.9).

Given an initial estimate of $\vec{W}$ Eqn. (4.27) and Eqn. (4.28) can be solved as convection equations for S and H. Then p and ρ may be evaluated from Eqn. (4.16). The convection Eqns. (4.23)-(4.26) can be solved for the adjoints λ, μ, η and ζ. This determines $\vec{Q}$ via Eqn. (4.18). The potential Φ may then be determined from Eqn. (4.22), i.e.

$$\vec{\nabla}\cdot(\rho\,\vec{\nabla}\,\Phi) = -\,\vec{\nabla}\cdot\vec{Q} \tag{4.29}$$

If the flow is known to be isoenergetic, then H can be set equal to H_∞ and we can delete Eqn. (4.28) and Eqn. (4.26) from the system. One can proceed similarly for isentropic flow. By choosing λ and μ to be zero at upstream infinity we guarantee that $\vec{Q}$ exists only where vorticity is present. Hence wherever potential flow exists,

Φ alone defines the flow. Euler flow can be interpreted as potential flow with field sources Eqn. (4.29) which exist only in regions having vorticity and whose strengths are determined from convection Eqns. (4.23)-(4.28).

4.2.2 Problems with Euler Equations

Although we have cast the Euler equations as a generalization of the full potential equation, the introduction of convection equations creates a considerable number of numerical problems. In this section we discuss the problems we feel must be addressed and to some extent resolved before a considerable investment is made in a production full configuration Euler code. (The results shown below are obtained from a special test code written for the Euler equations)

The first problem concerns false production of a convected quantity. As a rule, we want to solve the conservation equations (4.1)-(4.3) in conservative form, not simply to capture discontinuities correctly, but to calculate accurate total forces and moments for large, complex configurations where truncation errors are hard to control. However, if we solve the Euler equations in conservative form, convection equations such as (4.8) will effectively have non-zero field sources on the right hand side due to truncation errors. This means that entropy may increase or decrease along a streamline when it should remain constant. The error is not locally confined, since false entropy which is generated upstream will convect downstream. In many current codes entropy production is responsible for poor drags and boundary layer matching as well as premature separation.

Even if Eqn. (4.8) were to be solved directly, numerical diffusion errors would still create problems. Convection operators such as $\vec{W} \cdot \vec{\nabla}$ all have inherent diffusion due to truncation errors. For grids used for inviscid modeling this numerical diffusion is orders of magnitude greater than that produced by viscous terms of the Navier-Stokes equations, hence nonphysical results are possible. The numerical diffusion is greatest when crossflow gradients are largest, e.g. at slip surfaces. To illustrate the problem we consider channel flow over a rectangular bump. A numerical solution was obtained using a rectangular grid ad hoc test code. In Figure 4.1 we show the results of convecting a smooth distribution of entropy at the entrance using a fairly good upwind discretization of the $\vec{W} \cdot \vec{\nabla}$ operator. The isentropic curves correspond closely to streamlines. This is true even for the bottom streamline which passes through regions of stagnation as well as large expansion. In Figure 4.2 we show results in the case of a discontinuous initial distribution of entropy. Here a considerable amount of diffusion takes place even on streamlines which lie in a region of relatively uniform flow. Clearly such diffusion must be eliminated if one wishes to calculate the effects of wing wakes on downstream components of the configuration. One can use non-diffusive numerical schemes which require every value of entropy to be precisely equal to some upstream value in the absence of legitimate dissipation. In Figure 4.3 we show the results of using such a scheme. Obviously there are no diffusive errors. However displacement errors are still possible when using such a scheme although they are rather small for this particular case. The major problem is

that a non-diffusive scheme is difficult to implement in a flux conservative formulation, and it is this problem which will require considerable effort to solve. There are a variety of possible approaches which introduce additional degrees of freedom so that the convection and flux conservation equations may be solved simultaneously. The Clebsch formulation is one such approach, and a displaced location for convected unknowns is another. These approaches will also solve the false entropy production problem as well.

A third problem with the Euler equations concerns uniqueness. There are certain situations where an Euler solution cannot exist without separation and closed streamlines[83]. Obviously the level of the convected quantities is indeterminate on closed streamlines. In fact the size of the separation region itself will depend on what value one's program happens to assign to the convected quantities. Thus quantities such as drag and lift will turn out to be somewhat arbitrary. There is not much one can do about this problem except try to eliminate false entropy production so that one achieves an unseparated solution when it exists. If such a solution does not exist then probably one should strive for a solution which minimizes the extent of the separation region.

A fourth problem with Euler equations concerns vortex separation (or swirl generation). The Euler equations do not contain enough physics to predict the location of separation lines or strength of separation except in special cases such as sharp edges. Thus one must be able to effect separation on the basis of outside knowledge. The first task is to clean up false entropy and swirl production so that premature separation does not take place. Secondly one must be able to specify the separation line and type of separation directly. Pure vortex separation is achieved by specifying a source for swirl only. (Allowing entropy increases will lead to contaminated vortex separation.) The strength of the swirl sources must be determined by a Kutta condition.

A final problem concerns vortex instabilities and related non-existence. Current literature[84] seems to indicate that the Euler equations have a legitimate solution only in special cases (e.g. potential flow). Vorticity seems to collect in unstable cores with increasing concentration, and blowup may occur in finite time. The blowup can be prevented by numerical diffusion. However in attempting to eliminate excess numerical diffusion for other reasons we may encounter vortex instabilities, and then the question becomes how much numerical diffusion is correct. This can be determined only by considering the full Navier-Stokes equations.

4.3 WAKE CAPTURING

We believe that if we can implement a good wake capturing scheme in TRANAIR then we will be able to handle 85% to 95% of the cases that an Euler solver could handle with much less risk, development cost, and run cost. This is due to the fact that most inviscid problems of interest in full configuration analysis really involve regions of potential flow separated by vortex sheets. These regions may possess different total pressures and temperatures, but we have already demonstrated the ability of TRANAIR to account for such effects (see Section 5.2.6). It is true that shocks

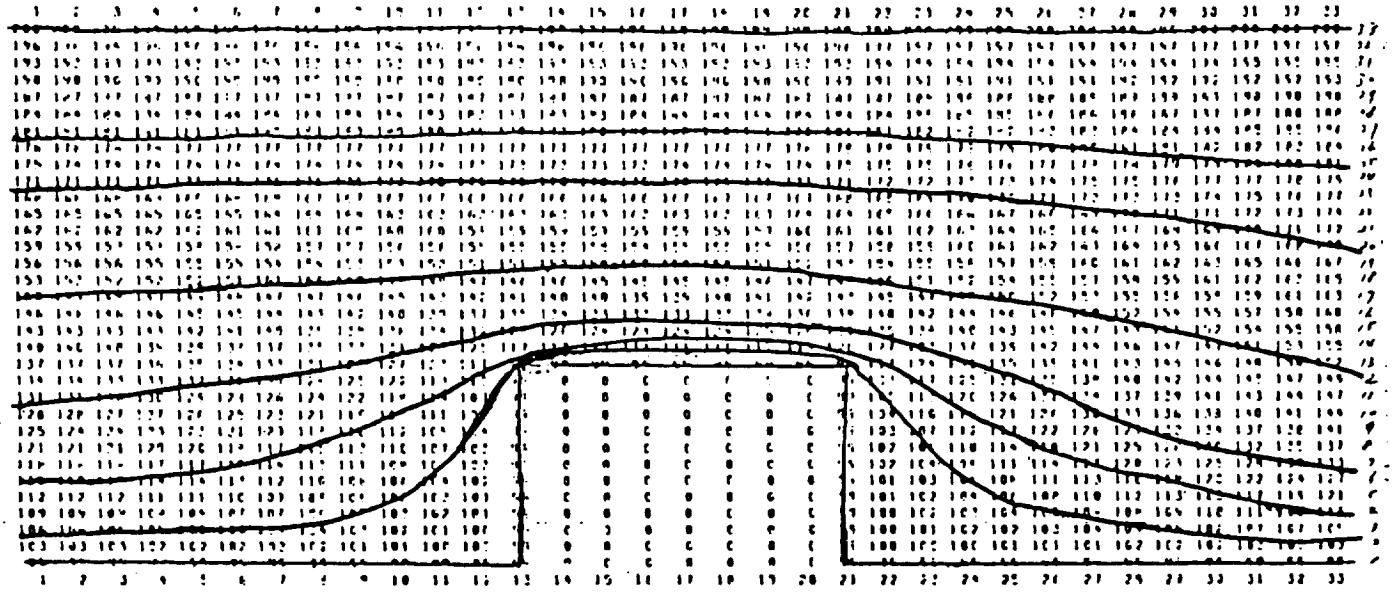


Figure 4.1: $\hat{W} \cdot \vec{\nabla} S = 0$ (Good Upwind Discretization Scheme) Smooth Inflow Distribution

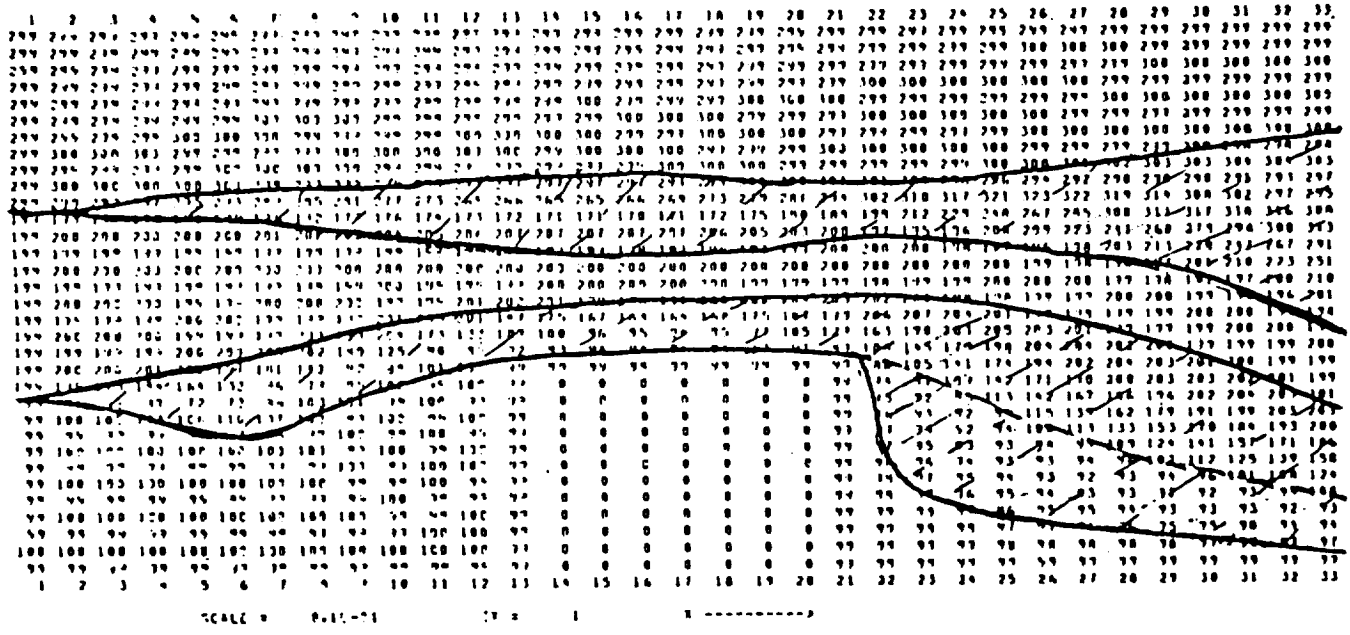


Figure 4.2: $\hat{W} \cdot \vec{\nabla} S = 0$ (Good Upwind Discretization Scheme) Discontinuous Inflow Distribution

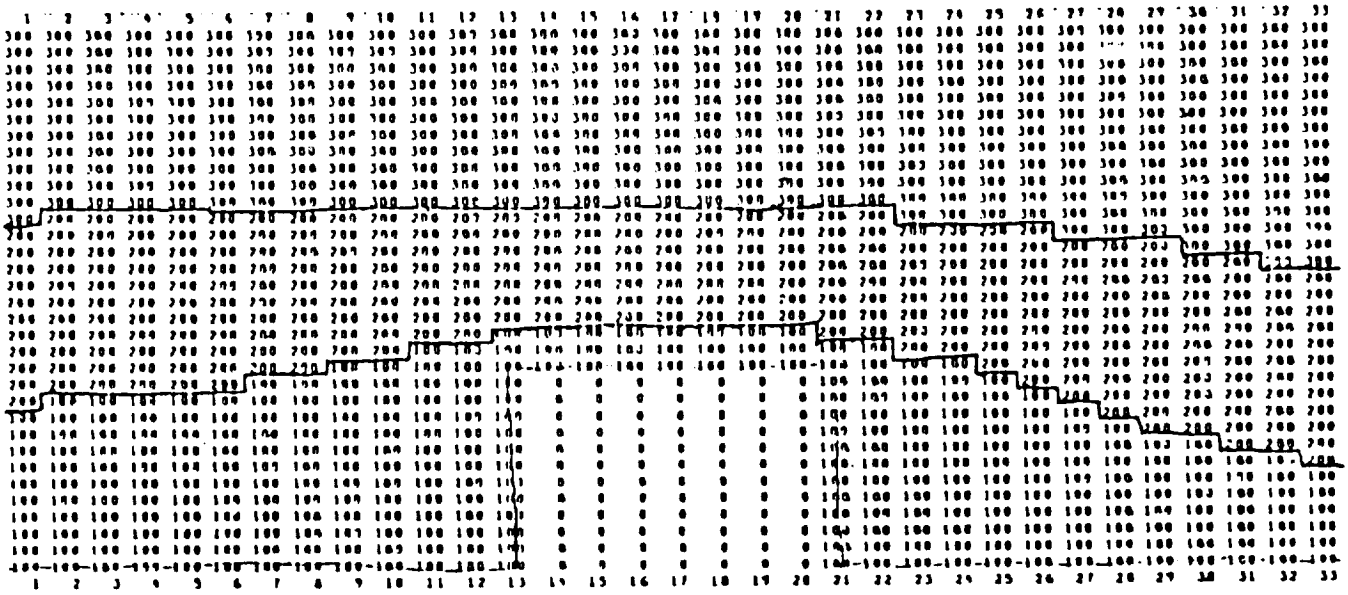


Figure 4.3: $\hat{W} \cdot \vec{\nabla} S = 0$ Non-Diffusive Scheme, Every Value of Entropy Equal to Some Upstream Value. No Interpolation Allowed.

generate volume vorticity. This vorticity is generally negligible except for extremely strong shocks, hence in most cases the Hafez correction [77] can probably produce the correct shock strength. It is true that TRANAIR cannot model volume swirl effects, but such effects could be modeled by collecting the swirl into vortex sheets and then employing the wake capturing feature described below. Hence, it is clear that the addition of a wake capturing algorithm will make TRANAIR rival many Euler codes in capability, while offering the advantage of reliability, efficiency and the ability to analyze extremely complex geometries without a great deal of user effort.

A variety of approaches for capturing wakes, short of solution of the full Euler equations, are possible. Unfortunately, many of them suffer from the same problem that was discussed in the previous section, i.e., excessive numerical diffusion arising from the discretization of convection equations. It is true that the adaptive grid capability might allow us to ignore the problem by concentrating sufficient grid in wakes to keep diffusion under control. However, a rough calculation shows that to convect vortex cores and sheets from the wing to tail with sufficient accuracy to be able to predict accurate tail loads would require at least triple the grid used to solve the problem with fixed wakes. The cost would currently be prohibitive. We have therefore been exploring compromises.

There are many arguments to consider in developing a wake capturing algorithm, some of which have been mentioned above. We have finally arrived at what we believe is a reasonable compromise between accuracy and user effort. This algorithm is based on a novel method developed and communicated to us by S. S. Desai[86] which combines vortex tracing methods with a non-linear full potential algorithm. The authors assume separation lines are specified and then emit discrete vortex filaments from these lines. These vortex filaments are aligned with the mean flow which is determined by combining the velocity induced by these vortices with the velocity computed on the full potential grid. This method often works quite well, but occasionally has a few problems. First the computation of the velocity induced by the vortex filaments is expensive, and second, this velocity is highly singular, resulting in vacuum conditions near each filament.

We are currently analyzing several modifications to this method. First we note that vortex filaments are equivalent to the edges of constant strength doublet panels. By employing linearly varying doublet panels instead, the $1/r$ singularities can be eliminated. Moreover, by interpreting the doublet panels as jumps in potential, one can take their influence into account through local jump conditions rather than through influence coefficients. The net effect of these two modifications is equivalent to a method whereby the positions of the current wake type-18 networks are updated so that each doublet panel side edge is aligned with the local mean flow direction. (We have checked that this condition is still applicable when the total pressure and temperature are different on the upper and lower sides of the doublet panels. Here one must calculate the mean flow direction using upper and lower mass flux vectors scaled by appropriate factors based on total pressure and temperature.)

At the moment we are accounting for the effect of wake panels on the local flow by incorporating them in the local D-region operators. This limits the generality of wake shape by requiring that wake panels cannot cut themselves or any portion of the

vehicle boundary. Moreover, D-region operators must be recomputed every time the wake position is updated. It would be better to "capture" the discontinuities induced by the doublet panels by computing their contribution to the field operators on-the-fly, using methods similar to those used in capturing thin layers in EM TRANAIR[40]. Such a procedure would, for example, allow a wing wake sheet to cut a tail without having to separate the wake sheet into two pieces.

The main advantage of the technique just described is that wake diffusion is virtually eliminated. Moreover, one does not have the expense of adding extra unknowns or derivatives everywhere in the flow field just to account for the possible existence of a wake. In one sense the method could be called "wake fitting". However the analogy to "shock fitting" doesn't really hold. Only the separation line really needs to be specified by the user. This is reasonable since separation is basically a viscous phenomenon. Once the sheet gets started there is no problem associated with 'recognizing' a wake as there is in shock fitting. In fact the method is much more closely related to the Clebsch decomposition. Here the μ parameter is precisely the doublet strength in the wake and the gradient of the λ lambda parameter corresponds to the normal vector of the doublet sheet.

4.4 BOUNDARY LAYER

In the vast majority of flow cases of practical interest the effects of viscosity are confined to a boundary layer next to the configuration surface. The influence of a boundary layer on the outer inviscid flow can be of major significance in some instances. One instance is where boundary layer separation produces a vortex sheet extending out into the inviscid flow field. Another instance is where shock-boundary layer interaction effects cause substantial thickening of the boundary layer and a correspondingly large modification of the effective configuration surface as seen by the outer inviscid flow. The latter effect is often of great importance in transonic analyses. In Figure 4.4 we show an analysis performed by Boeing's A488 code[87] on the 747-200 wing at Mach 0.86 and at 2.70 degrees angle of attack. The code was run with and without boundary layer coupling and the results were compared to experiment. The coupled results are in much better agreement with the experimental results, and the primary effect of the boundary layer appears to be a weakening and upstream displacement of the normal shock. For some wings the effect is less pronounced, but one cannot know this without doing the actual boundary layer analysis.

It would be a fairly straightforward task to couple the boundary layer code in A488 to TRANAIR. However, boundary layer codes often tend to be the weak link in a flow analysis. Transition models, turbulence models, and shock-boundary layer interaction models are all very ad-hoc in nature. There is not much that can be done about this. In addition, coupled transonic/boundary layer codes often have convergence problems due to the coupling itself. The input to most boundary layer codes is the inviscid pressure distribution and the attachment line. The boundary layer code proceeds in a marching fashion to generate boundary layer thickness for delivery back to the inviscid code, and the inviscid code generates a new solution.

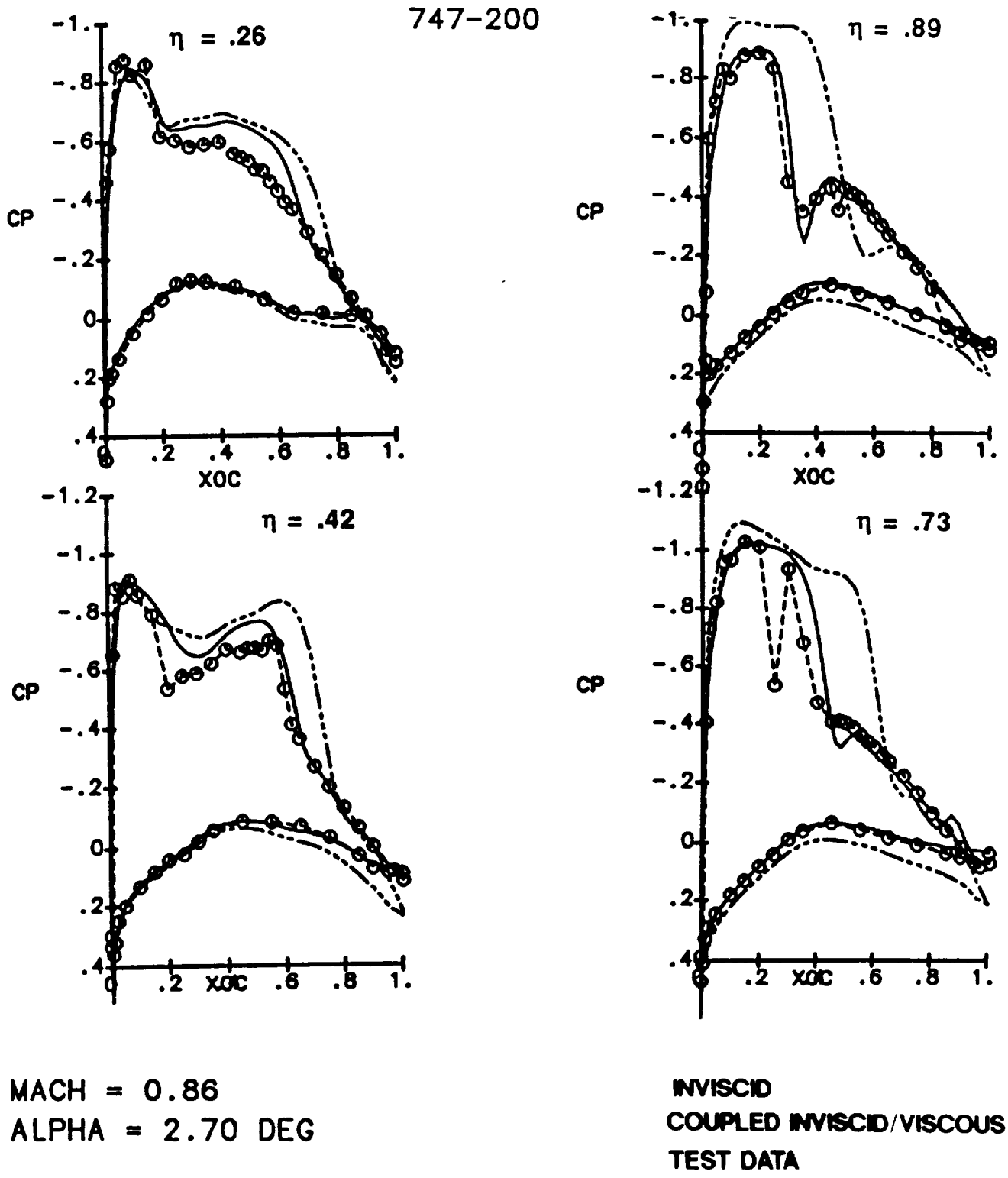


Figure 4.4: Transonic Analysis Demands Viscous Coupling

This procedure is repeated until convergence, which is not always achieved. Moreover, premature separation can halt the boundary layer marching procedure.

Due to the use of a sparse direct solver TRANAIR could be coupled to a boundary layer code directly. That is, the boundary layer equations could be treated in the same manner as the pressure equations (see Section B.3). Moreover, because of the direct nature of the solution, marching is no longer necessary (although the equations would be arranged in marching order to minimize fill-in). Hence, it would be possible to allow the introduction of elliptic terms, resulting in, e.g., the thin layer Navier-Stokes equations.

4.5 DESIGN AND OPTIMIZATION

TRANAIR currently has a rudimentary sequential inverse design capability. It allows the user to specify pressure coefficient at upper surface corner points of a thick surface network, or jump in pressure coefficient and thickness slope at corner points of a thin surface network. In the first case the network surface is to be relofted parallel to the upper surface mass flux vector. In the second case the surface is to be relofted parallel to the average mass flux vector. No relofting capability has been attached to TRANAIR, as such a capability is strongly case dependent and intimately tied to one's geometry generation system.

The procedure just described is similar to the cycled boundary layer coupling described in the previous subsection. It can often be effective, but is certainly not robust and may require intervention by an expert user. We again prefer a more direct approach based on the use of the sparse solver. In such an approach the parameters describing variations in geometry would be combined with flow unknowns and the whole system including pressure specification and impermeability conditions would be solved as a directly coupled system.

Development of a directly coupled inverse design program would represent a major step towards a full optimization capability. Here an actual payoff function would be minimized with respect to a set of controls (geometry perturbation parameters), subject to inequality constraints on these controls as well as the state equations (full potential equation).

Chapter 5

CONCLUSIONS

A new approach to solving the full potential equation about arbitrary three-dimesional geometries has been presented. This approach has been implemented in a computer code called TRANAIR. A wide variety of subsonic, transonic and supersonic results have been presented. They indicate that TRANAIR has made substantial progress towards the objective of offering aerospace vehicle designers a reliable, general purpose, full configuration flow analysis tool that is relatively easy to use. In particular, these results show that it is indeed possible to eliminate the costly and time consuming process of generating a surface fitted grid while maintaining the ability to capture small scale flow details accurately. Further work to improve TRANAIR and extend its domain of applicabilty has been discussed.

Chapter 6

Acknowledgements

This work was supported in part by NASA contract NAS2-11851 and the Boeing Independent Research and Development Funds. We wish to thank Prof. K. D. Lee of the University of Illinois for his early contributions to this project. In addition, we acknowledge the contributions of L. B. Wigton of the Boeing Commercial Airplanes for work described in Appendix C. We acknowledge the contributions of R. H. Burkhart of the Boeing Computer Services for the work described in Appendix D. We acknowledge the contributions of B. L. Eversen of the Boeing Computer Services for the work on many program libraries used in TRANAIR. We also acknowledge the help we received from Edward Tinoco and Allen Chen and Margaret Curtin also from Boeing Commercial Airplanes in running some of the test cases.

Appendix A

OCT-TREE DATA STRUCTURES

A.1 DATA STRUCTURE ORGANIZATION

A compact data structure which contains essentially all the information regarding the refined grid has been developed. It allows the TRANAIR code to concentrate many small boxes in areas where greater solution detail is needed and fewer and larger boxes in areas where less solution detail is desired. While usually refered to as '*the oct-tree*', the data structure is actually a forest of oct-trees where each oct-tree root is a box in a uniform, regularly indexed grid. The data structure allows efficient extraction of a variety of information, such as the location of nodes and element centroids, box size, box level, node indices, box adjacency, and identity of boundary boxes.

A.1.1 Base Grid

The global grid (described in Section 2.3.3) specified over the computational domain is uniformly derefined to obtain the *base grid.* Each base grid box becomes the root of an oct-tree. All the descendent boxes of each base grid box physically lie within that base grid box.

A.1.2 Oct-Trees

Each box in the data structure can be recursively subdivided (refined) into eight similar boxes. The hierarchy of boxes formed in this process is known as an *oct-tree.* The oct-tree data structure represents a parent-child relationship between a box and the sub-boxes formed by its subdivision and also the sibling relationship between the sub-boxes.

Some restrictions are placed on the refinement to minimize data structure size and to simplify the problem. First, boxes are refined by subdivision into exactly eight equally sized sub-boxes. This greatly reduces the data structure size by eliminating the need to store box centroids and sizes. It also has the effect of keeping the aspect ratio of all boxes equal. Box centroids and sizes are derived from the box's position in

the oct-tree hierarchy. Second, no two face or edge neighbors in a "legal" refined grid differ by more than one level, (see Figure A.1), which greatly reduces the solution computation complexity.

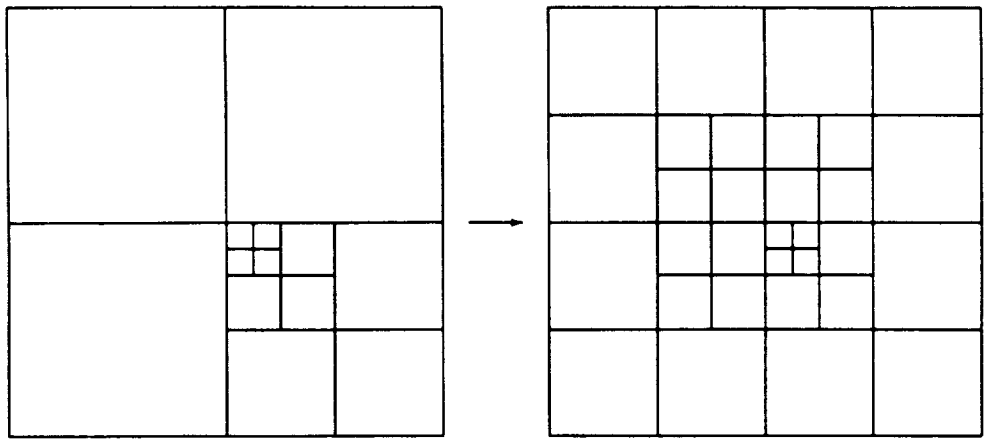


Figure A.1: Grid "Legalization" Example.

The basic oct-tree data structure is based on boxes. However, it has been extended to accommodate nodal information. A *node* is located at a corner point of one or more boxes. Nodes are indexed by assigning a box to the node at its lower-left-near corner. To account for all nodes at refinement interfaces, a pseudo-refinement is performed (Figure A.2). Pseudo-refinement creates boxes (called *pseudo-boxes*) that are assigned to nodes, but are not used as finite elements in the solution process.

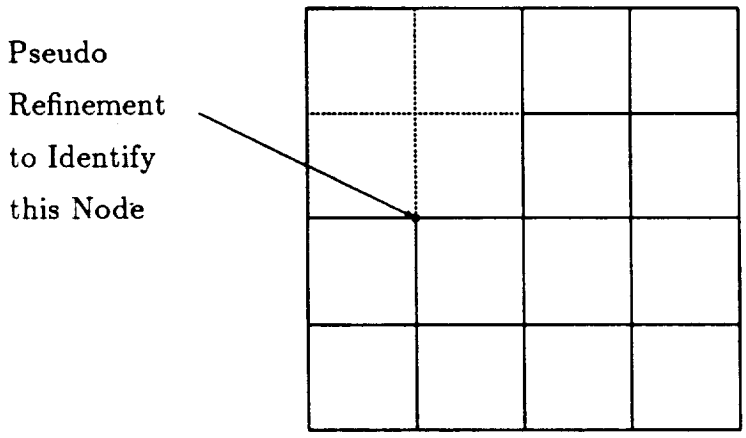


Figure A.2: Pseudo-refinement to Represent the Nodes

A.1.3 Terminology

- *B-Box* (Base Grid Box): Any of the base grid boxes in the uniform, regularly indexed grid derived from the *global grid*. *Base grid* boxes are the roots of each of the oct-trees and have no parents.
- *O-box*: Any box in the oct-tree.
- *R-Box* (Red Box): Any unrefined box. An R-Box can contain a finite element trial function.
- *G-Box* (Green Box): A pseudo-box created so that all nodes at refinement interfaces are associated with a box. Some G-boxes can lie outside the computational domain to define the nodes on the boundary of the computational domain.
- *U-Box*: Any box (or pseudo box) whose lower-left-near corner is associated with a node.
- *T-box*: An *R-box* that intersects the boundary.

A.2 DATA STRUCTURE REPRESENTATION

The data structure used in TRANAIR to describe oct-trees is a modification of that described by Samet [57]. The data structure is divided into six areas: the header, the base grid descriptor, the refinement family, a stack, a boundary box map, and refinement pointers. A global overview of the data structure array is shown in Figure A.3.

A.2.1 Header

The header area is fixed in size and location. It maintains a variety of information about the data structure including the locations of its various components and certain statistical information about the data structure and the grid.

A.2.2 Base Grid Descriptors

The header area is followed by a base grid descriptor area. This area is divided into two regions, the *base grid pointers* and the *U-box accumulators* for the base grid. The two regions are laid out as parallel arrays of the size of the base grid. Each element of the base grid pointer region identifies the location of a *refinement family* for a particular base grid box. A zero value here implies that the base grid box is not refined. The U-box accumulators are the number of U-boxes encountered in the data structure during a sequential traversal of the data structure.

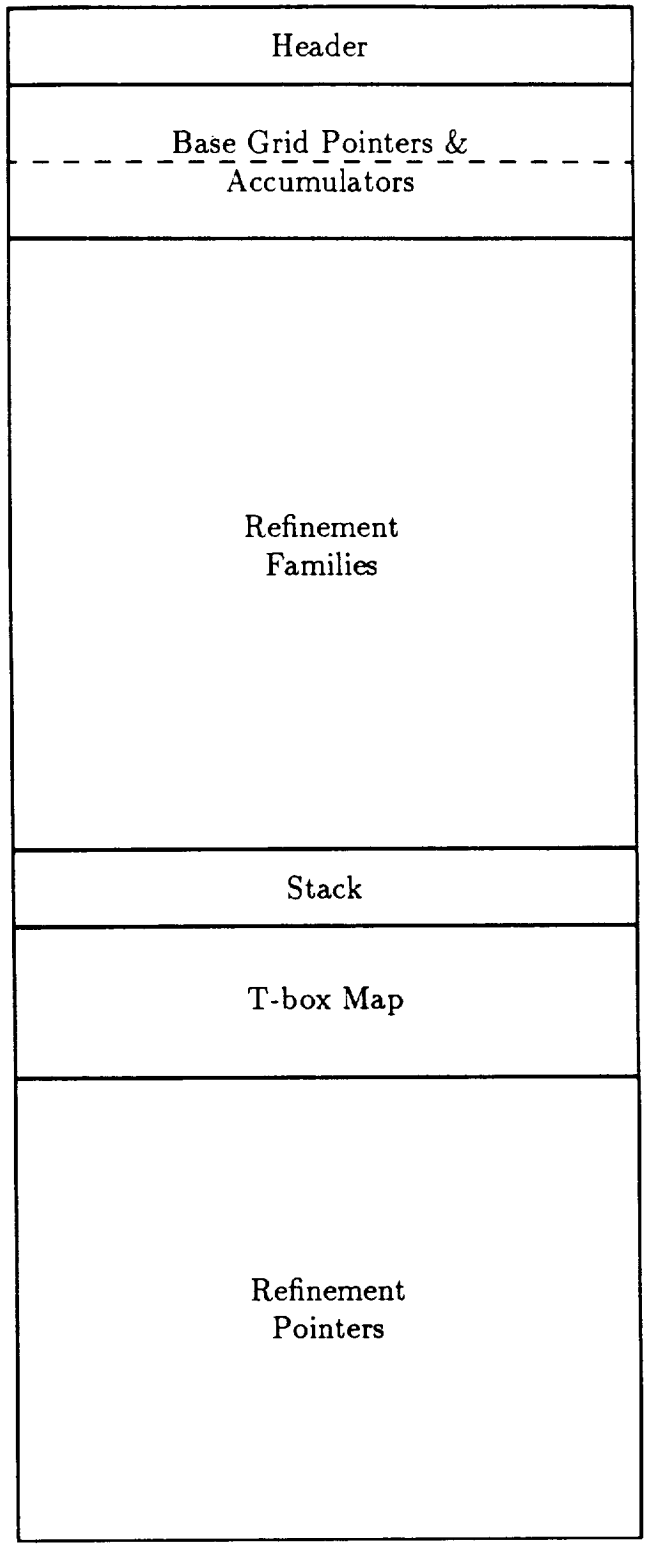


Figure A.3: The Overview of the Oct-tree Data Structure

A.2.3 Refinement Families

The base grid descriptor area is followed by a series of *refinement families*. A refinement family describes a subdivision of a box into eight smaller boxes. A refinement family is shown in Figure A.4. Each refinement family data block consists of five integer fields:

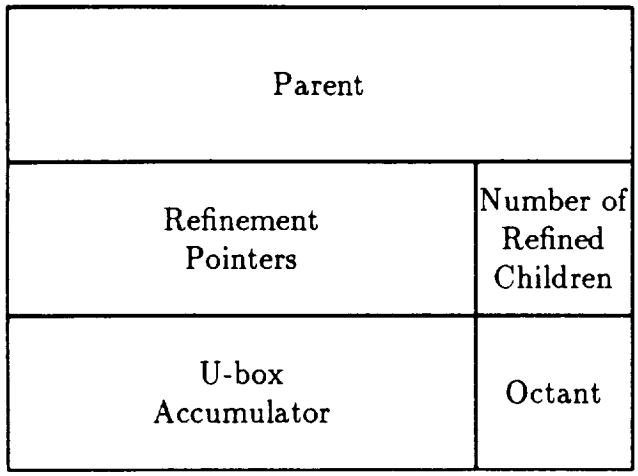


Figure A.4: A Refinement Family

- The first field contains the address of the parent. The parent pointer is used to facilitate upward traversals in the tree.
- The second field contains the location of a data block that describes the children of this refinement family. A zero value in this field indicates that none of the eight children are refined.
- The third field describes the number of refined children contained in the block referenced by the second field.
- The fourth field is the U-box accumulator. It describes the number of U-boxes encountered in the data structure during a sequential traversal of the data structure.
- The fifth field is the octant of this refinement family in the refinement of its parent.

Because the third (number of refined children) and fifth (octant) fields contain small values, it is reasonable to compress these values into other data fields to conserve memory. As a result, only three integers are used to store the refinement families.

The first integer is equal to the first field. The second integer contains the second and third fields of the data structure. The values for these fields are given by:

```
FIELD #2 = (INTEGER #2) / 16
FIELD #3 = ABS((INTEGER #2) MOD 16)
```

The third integer contains the fourth and fifth fields. The values for these fields are given by:

```
FIELD #4 = (INTEGER #3) / 16
FIELD #5 = ABS((INTEGER #3) MOD 16)
```

As boxes are refined, refinement families are appended to this list. This area grows towards the end of the total data structure.

A.2.4 Scratch Stack

The refinement family area is followed by a small scratch stack area that is used to record traversal paths. The stack area is not used until *after* all refinements have been made and the refinement family list has stopped growing.

A.2.5 T-box Map

The stack area is followed by a T-box mapping vector. The mapping describes which O-boxes have a non-empty intersection with the boundary. The map is constructed as an ordered (ascending) array of those O-boxes who intersect the boundary. The creation of this map is the final step in the generation of the oct-tree data structure.

A.2.6 Refinement Pointers

The final area in the data structure contains the *refinement pointers*. The refinement pointers describe the addresses of the refinement families for the children of refinement families. This area is composed of variable size blocks. Each block is composed of a set of number pairs that describe the address of a refinement family, and the octant that refinement family lies in. An example refinement block is shown in Figure A.5. Because the octant value is small, the octant and pointer values are stored together in one integer. The pointer and octant values are given from a data value as:

```
POINTER = (Data Value) / 16
OCTANT = ABS((Data Value) MOD 16)
```

Additionally, a *back pointer* is provided to facilitate oct-tree construction. The back pointer contains the address of the refinement family whose refinements are being described by this block. This pointer, like the others is stored with an octant value. Back pointers are assigned and octant value of zero to distinguish them from sibling refinement pointers. The back pointers are removed to conserve memory after the oct-tree is constructed. A flag in the header field indicates the presence of the back pointers.

The refinement pointer area grows from the end of the data structure toward the refinement families. When insufficient space remains for growth the data structure size is enlarged within the code so that expansion can occur.

Pointer	Octant
Pointer	Octant
Pointer	Octant
Back Pointer	0

Figure A.5: A Refinement Pointer Block

A.3 MAJOR ALGORITHMS

This section describes the major algorithms used to manipulate and interrogate the oct-tree data structure. The description of the algorithms are sketches and are not intended to be exhaustive explanations.

A.3.1 Data Structure Modification

Refining a Box

When a given (unrefined) O-box is refined, a refinement family is added to the refinement families area of the data structure. The parent of the new refinement family is the O-box being refined. The octant of the box being refined is defined by:

```
OCTANT = ((O-box - NXYZB - 1)) MOD 8 + 1
```

where **NXYZB** is the number of base grid boxes. The refinement pointer block of the O-box's parent refinement family is modified so that the **OCTANT** octant of the pointer block contains the address of the new refinement family.

Creating the O-box/U-box Mapping

The O-box/U-box mapping is created by storing in each refinement family an accumulation of the number of U-boxes encountered in the boxes defined by families that precede it in the list.

Creating the T-box/O-box Mapping

The T-box/O-box mapping is implemented as an ordered (ascending) list of O-boxes.

A.3.2 Data Structure Interrogation

Finding Refinement Family

The index of the refinement family defining an O-box is given by:

```
FAMILY = ((O-BOX - NXYZB - 1 ) / 8) + 1
```

where NXYZB is the number of base grid boxes.

Determining Box Number From Refinement Family Index

The O-box number of a refinement family can be found by:

```
OBOX = NXYZB + 8 * FAMILY + OCTANT
```

where NXYZB is the number of base grid boxes and OCTANT is the octant of this family in the refinement of its parent.

Finding Box Centroid and Size

Box centroids and sizes are calculated by ascending the oct-tree hierarchy to the base grid root. At each level reached in the hierarchy, the box centroid (initially *(0,0,0)*) is moved towards the parent box's centroid by examining the current octant and number of levels traversed. When the base grid box is reached, the calculated centroid is translated to its center. The size of the box is equal to

```
1 / ( 2 ** LVL ) * BGSIZE
```

where LVL is the number of levels traversed and BGSIZE is a vector describing the dimensions of the base grid box.

Finding Child Boxes

A given box has either eight or no children. If refinement family for the box exists then it has eight children. Otherwise, the box is unrefined (has no children). Child O-box numbers are given by:

```
OBOX = NXYZB + 8 * FAMILY + OCTANT
```

where NXYZB is the number of base grid boxes and OCTANT is number lying between one and eight.

Finding Neighboring Boxes

Neighboring boxes are found by traversing up the oct-tree hierarchy until either an ancestor common to both the box and the neighboring box is found, see Figure A.6. The common ancestor is found if the last parent in the pedigree is on the complementary side of the child (i.e. if the south neighbor is desired then the parent should be on the north side of its child). A downward traversal along a path complementary to the upward path is performed. If the root is reached and the ancestor is yet to be found then the neighboring box lies in a different tree in the oct-tree forest. The neighboring tree is the base grid box lying in the desired direction. For a legal tree, there can be zero, one, or four neighbors in any face direction and zero, one, or two neighbors in any edge direction. In Figure A.6 the traverse path for finding the *north* neighbor of the box 1 is shown as an illustration.

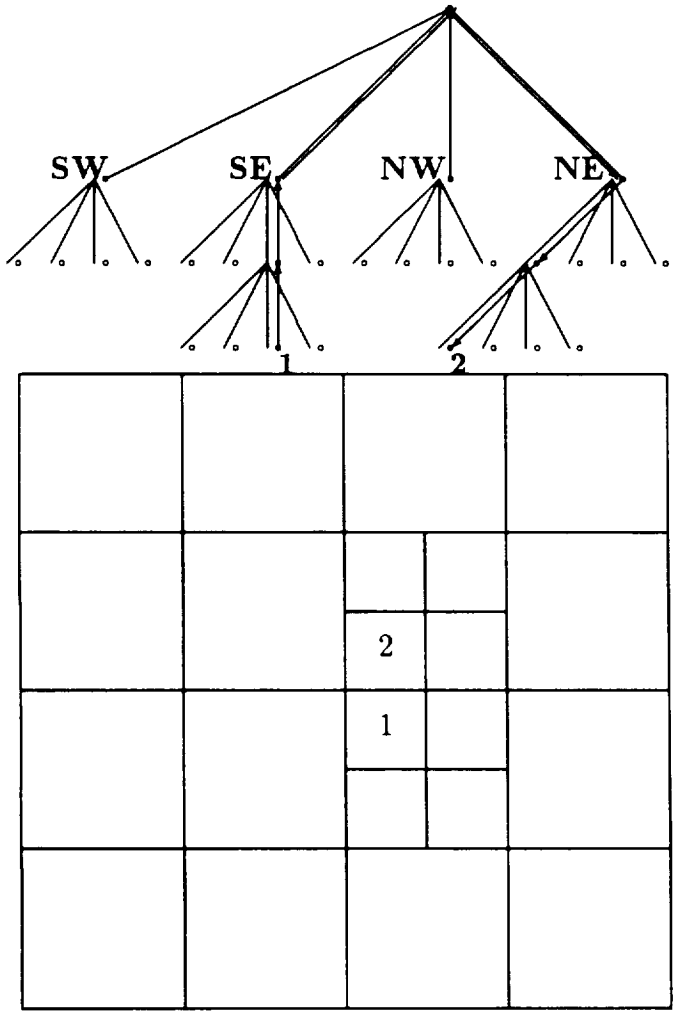


Figure A.6: Finding Neighboring Boxes.

Mapping O-box and U-box Indices

To find the U-box index of an O-box, the refinement family of the O-box must be found. If the O-box is refined and the refinement is not a pseudo-refinement, then it is not a U-box, otherwise, the U-box accumulator is recovered from the refinement family. The U-box accumulator is then decremented by one for each sibling whose octant is greater than that of the O-box, and who is not refined or who is a pseudo-refinement.

The O-box index of a given U-box can be determined by performing a binary search on the (ordered) list of U-box accumulators to find the refinement family whose U-box accumulator is closest to that of the given U-box. The U-box accumulator is incremented by one for each child who is either unrefined or who is a pseudo-refinement until the U-box accumulator is the desired U-box. The O-box is the last child who caused the U-box accumulator to be incremented.

Mapping O-box and T-box indices

To find the O-box associated with a particular T-box index simply involves retrieval of the value from the T-box map. A binary search is performed to recover the T-box index given an O-box.

Finding the U-Box Containing a Point

To find the U-box containing a point in the computational volume, determine the base grid box containing the point. Traverse the oct-tree rooted at the base grid box by choosing, at each level, the child box that contains the point. The traversal terminates when a leaf node of a pseudo-refinement is found.

Appendix B

OPERATOR DEFINITION

In this appendix we describe how the various operators (discrete equations) in TRANAIR are defined and constructed.

B.1 IMPLEMENTATION

In this section we discuss the way in which the Bateman principle is used to create operators at each grid point. The departure point is Eqn. (2.11) which is repeated here

$$\delta J = \int_{\Omega} - \vec{W} \cdot \delta \vec{V} \, d\Omega \tag{B.1}$$

We take Ω as some subregion of the grid box shown in Figure B.1. First we define perturbation potential ϕ as

$$\phi = \Phi - \vec{V}_{\infty} \cdot \vec{r} \, , \; \vec{V}_{\infty} = (U_{\infty} \, , \vec{V}_{\infty} \, , W_{\infty}) \tag{B.2}$$

and then rewrite Eqn. (B.1) as

$$\delta J = \int_{\Omega} -\rho \;\; (V_{\infty} + \vec{\nabla}\phi) \cdot \;\; \delta\vec{\nabla}\phi \, d\Omega \tag{B.3}$$

We define ρ_o as $\rho({q_o}^2)$ in the following equation

$$\rho = \rho_{\infty} \left[1 + \frac{\gamma - 1}{2} M_{\infty}^2 (1 - \frac{{q_o}^2}{q_{\infty}^2}) \right]^{\frac{1}{\gamma - 1}} \tag{B.4}$$

where q_o is the value of q at the centroid of Ω. We then make the approximation that

$$\delta J \approx -\rho_o \int_{\Omega} (\vec{V}_{\infty} + \vec{\nabla}\phi) \cdot \;\; \delta\vec{\nabla}\phi \, d\Omega \tag{B.5}$$

This approximation is valid in incompressible flow where ρ is constant. If the basis function for ϕ in Ω is linear then $\vec{V}$ is constant in Ω and Eqn. (B.5) is still valid. However the basis function for ϕ is actually trilinear and strictly speaking $\vec{V}$ varies in Ω, but because we evaluate ρ_o at the centroid of Ω, Eqn. (B.5) and (B.3) are the same to second order.

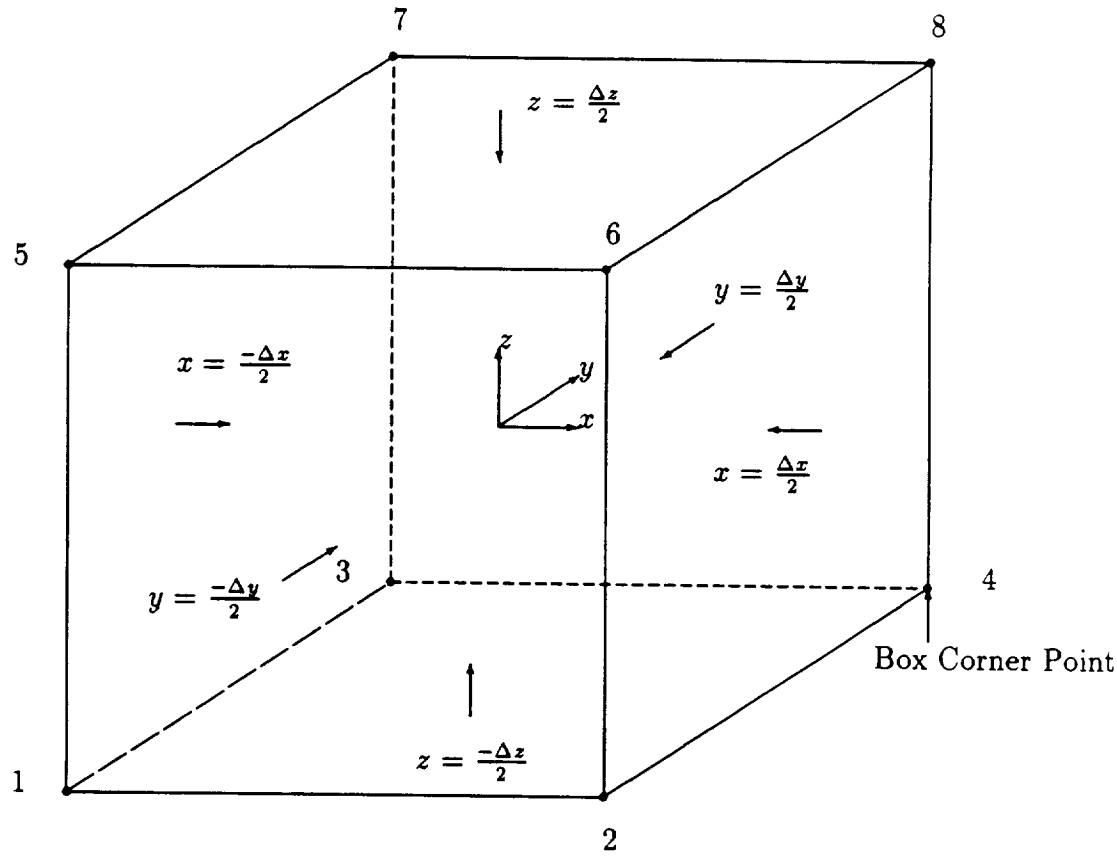


Figure B.1: Grid Box

The basis function for ϕ in the grid box of Figure B.1 is defined as

$$\phi = \phi_o + \phi_x x + \phi_y y + \phi_z z + \phi_{xy} xy + \phi_{yz} yz + \phi_{zx} zx + \phi_{xyz} xyz \tag{B.6}$$

where the origin for the x, y, z coordinate system is in the center of the box. Note that this expression has eight unknown coefficients. These coefficients are chosen so that ϕ exactly fits the eight values ϕ_1, ϕ_2, ϕ_3, ϕ_4, ϕ_5, ϕ_6, ϕ_7, ϕ_8 at the box corners. To express this fact in compact form we define

$$\xi = \frac{x}{\Delta x}, \eta = \frac{y}{\Delta y}, \zeta = \frac{z}{\Delta z} \tag{B.7}$$

Note that for the box of Figure B.1, $| \xi | \leq \frac{1}{2}$, $| \eta | \leq \frac{1}{2}$, $| \zeta | \leq \frac{1}{2}$. We rewrite Eqn. (B.6) as

$$\phi = \phi_o + \phi_\xi \xi + \phi_\eta \eta + \phi_\zeta \zeta + \phi_{\xi\eta} \xi\eta + \phi_{\eta\zeta} \eta\zeta + \phi_{\zeta\xi} \zeta\xi + \phi_{\xi\eta\zeta} \xi\eta\zeta \tag{B.8}$$

where $\phi_\xi = \Delta x \phi_x$, etc... This can be rewritten as

$$\phi = \vec{b} \cdot \vec{\omega} \tag{B.9}$$

where

$$\vec{b} = (1, \xi, \eta, \zeta, \eta\zeta, \zeta\xi, \xi\eta, \xi\eta\zeta) \tag{B.10}$$

and

$$\vec{\omega} = (\phi_o, \phi_\xi, \phi_\eta, \phi_\zeta, \phi_{\eta\zeta}, \phi_{\zeta\xi}, \phi_{\xi\eta}, \phi_{\xi\eta\zeta}) \tag{B.11}$$

But for some matrix R,

$$\vec{\omega} = R\vec{\tau} \tag{B.12}$$

where

$$\vec{\tau} = (\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7, \phi_8) \tag{B.13}$$

Thus

$$\phi = \vec{b} \cdot R\vec{\tau} \tag{B.14}$$

The matrix R can be calculated by evaluating ϕ of Eqn. (B.9) at each of the eight corner points to obtain

$$\vec{\tau} = B\vec{\omega} \tag{B.15}$$

Here B is an 8x8 matrix whose rows are $\vec{b}$ evaluated at each of the eight corner points. From Eqn. (B.12) R is the inverse of B. One can use a computer to invert B, but it is more satisfying to combine rows of B in a judicious manner to deduce R. For example, by adding all the rows of Eqn. (B.15) together, we get $\phi_1 + \phi_2 + \phi_3 + \phi_4 +$

$\phi_5 + \phi_6 + \phi_7 + \phi_8 = 8$ from which we deduce the first row of R. R is the following matrix.

$$\begin{bmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \\ -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \end{bmatrix} \tag{B.16}$$

Defining

$$U = \phi_x \, , V = \phi_y \, , W = \phi_z \tag{B.17}$$

we have from Eqn. (B.14) that

$$U = \frac{1}{\Delta x}\vec{b}_\xi \cdot R\vec{\tau} \tag{B.18}$$

$$V = \frac{1}{\Delta y}\vec{b}_\eta \cdot R\vec{\tau} \tag{B.19}$$

$$W = \frac{1}{\Delta z}\vec{b}_\zeta \cdot R\vec{\tau} \tag{B.20}$$

Here

$$\vec{b}_\xi = (0 \, , 1 \, , 0 \, , 0 \, , 0 \, , \zeta \, , \eta \, , \eta\zeta) \tag{B.21}$$

$$\vec{b}_\eta = (0 \, , 0 \, , 1 \, , 0 \, , \zeta \, , 0 \, , \xi \, , \zeta\xi) \tag{B.22}$$

$$\vec{b}_\zeta = (0 \, , 0 \, , 0 \, , 1 \, , \eta \, , \xi \, , 0 \, , \xi\eta) \tag{B.23}$$

One can now rewrite Eqn. (B.5) as

$$\delta J \approx -\rho_o[\vec{a}_\infty \cdot R\delta\vec{\tau} + \vec{\tau} \cdot R^T C R\delta\vec{\tau}] \tag{B.24}$$

where

$$\vec{a}_\infty = \int_\Omega [\frac{1}{\Delta x}U_\infty \vec{b}_\xi + \frac{1}{\Delta y}V_\infty \vec{b}_\eta + \frac{1}{\Delta z}W_\infty \vec{b}_\zeta] \, d\Omega \tag{B.25}$$

and

$$C = \int_\Omega [\frac{1}{\Delta x^2}\vec{b}_\xi \vec{b}_\xi{}^T + \frac{1}{\Delta y^2}\vec{b}_\eta \vec{b}_\eta{}^T + \frac{1}{\Delta z^2}\vec{b}_\zeta \vec{b}_\zeta{}^T] \, d\Omega \tag{B.26}$$

One can rewrite Eqn. (B.25) as

$$\begin{aligned}\vec{a}_\infty = & \frac{U_\infty \Omega}{\Delta x}(0,1,0,0,0,\overline{\zeta},\overline{\eta},\overline{\eta\zeta}) \\ + & \frac{V_\infty \Omega}{\Delta y}(0,0,1,0,\overline{\zeta},0,\overline{\xi},\overline{\zeta\xi}) \\ + & \frac{W_\infty \Omega}{\Delta z}(0,0,0,1,\overline{\eta},\overline{\xi},0,\overline{\xi\eta})\end{aligned} \tag{B.27}$$

where Ω is the volume of the region Ω and the superscript bar denotes mean value, e.g.,

$$\overline{\eta\zeta} = \frac{1}{\Omega}\int_\Omega \eta\zeta d\Omega \tag{B.28}$$

Similarly we have

$$\begin{aligned}C = & \frac{\Omega}{\Delta x^2}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \overline{\zeta} & \overline{\eta} & \overline{\eta\zeta} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \overline{\zeta} & 0 & 0 & 0 & \overline{\zeta^2} & \overline{\eta\zeta} & \overline{\eta\zeta^2} \\ 0 & \overline{\eta} & 0 & 0 & 0 & \overline{\eta\zeta} & \overline{\eta^2} & \overline{\eta^2\zeta} \\ 0 & \overline{\eta\zeta} & 0 & 0 & 0 & \overline{\eta\zeta^2} & \overline{\eta^2\zeta} & \overline{\eta^2\zeta^2} \end{pmatrix} \\ + & \frac{\Omega}{\Delta y^2}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \overline{\zeta} & 0 & \overline{\xi} & \overline{\zeta\xi} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \overline{\zeta} & 0 & \overline{\zeta^2} & 0 & \overline{\zeta\xi} & \overline{\zeta^2\xi} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \overline{\xi} & 0 & \overline{\zeta\xi} & 0 & \overline{\xi^2} & \overline{\zeta\xi^2} \\ 0 & 0 & \overline{\xi\zeta} & 0 & \overline{\xi\zeta^2} & 0 & \overline{\zeta\xi^2} & \overline{\zeta^2\xi^2} \end{pmatrix} \\ + & \frac{\Omega}{\Delta z^2}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \overline{\eta} & \overline{\xi} & 0 & \overline{\xi\eta} \\ 0 & 0 & 0 & \overline{\eta} & \overline{\eta^2} & \overline{\xi\eta} & 0 & \overline{\xi\eta^2} \\ 0 & 0 & 0 & \overline{\xi} & \overline{\xi\eta} & \overline{\xi^2} & 0 & \overline{\xi^2\eta} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \overline{\xi\eta} & \overline{\xi\eta^2} & \overline{\xi^2\eta} & 0 & \overline{\xi^2\eta^2} \end{pmatrix}\end{aligned} \tag{B.29}$$

In general Ω is a subdomain of the box bounded by polygons consisting of parts of subpanels and faces of the box. The mean quantities in Eqn. (B.27) and (B.29) can in general be evaluated only by using complicated recursions best programmed on the computer; see Section 2.3. However there are some special cases for which hand calculations are possible. Let us assume that Ω is a small subregion of the box which is close to a corner, say corner 8. Then clearly the mean values on Ω are almost the same as the values at corner 8, e.g.,

$$\overline{\xi} = \overline{\eta} = \overline{\zeta} = \frac{1}{2} \, , \overline{\xi\eta} = \frac{1}{4} \, , \overline{\xi\eta\zeta} = \frac{1}{8} \tag{B.30}$$

Another case of interest is when Ω is a rectangular region, e.g., the whole box, the upper half, the lower tenth, etc. Let us assume that Ω is defined by $x_1 \leq x \leq x_2$, $y_1 \leq y \leq y_2$, $z_1 \leq z \leq z_2$. Then the term $\overline{\xi^{L-1}\eta^{M-1}\zeta^{N-1}}$ is given by

$$\begin{aligned}
&= \overline{\xi^{L-1}\eta^{M-1}\zeta^{N-1}} \\
&= \frac{1}{\Omega}\int_\Omega \xi^{L-1}\eta^{M-1}\zeta^{N-1}d\Omega \\
&= \frac{1}{\int_\Omega d\Omega}\int_\Omega \xi^{L-1}\eta^{M-1}\zeta^{N-1}d\Omega \\
&= \frac{1}{\int_{x_1}^{x_2} dx}\int_{x_1}^{x_2}\xi^{L-1}dx\frac{1}{\int_{y_1}^{y_2} dy}\int_{y_1}^{y_2}\eta^{M-1}dy\frac{1}{\int_{z_1}^{z_2} dz}\int_{z_1}^{z_2}\zeta^{N-1}dz \\
&= \frac{1}{\int_{\xi_1}^{\xi_2} d\xi}\int_{\xi_1}^{\xi_2}\xi_i{}^{L-1}d\xi\frac{1}{\int_{\eta_1}^{\eta_2} d\eta}\int_{\eta_1}^{\eta_2}\eta_i{}^{M-1}d\eta\frac{1}{\int_{\zeta_1}^{\zeta_2} d\zeta}\int_{\zeta_1}^{\zeta_2}\zeta_i{}^{N-1}d\zeta \\
&= \frac{1}{L\cdot M\cdot N}\cdot\frac{\xi_2{}^L-\xi_1{}^L}{\xi_2-\xi_1}\cdot\frac{\eta_2{}^M-\eta_1{}^M}{\eta_2-\eta_1}\cdot\frac{\zeta_2{}^N-\zeta_1{}^N}{\zeta_2-\zeta_1}
\end{aligned} \tag{B.31}$$

Case 1:

Now assume that Ω is the whole box, i.e.,

$$\xi_2 = \frac{1}{2} \, , \; \xi_1 = -\frac{1}{2} \, , \; \eta_2 = \frac{1}{2} \, , \; \eta_1 = -\frac{1}{2} \, , \; \zeta_2 = \frac{1}{2} \, , \; \zeta_1 = -\frac{1}{2} \tag{B.32}$$

Clearly if L, M or N is even, the mean value on the left of Eqn. (B.31) vanishes. The only non-zero mean values in Eqn. (B.27) and (B.29) are then

$$\begin{aligned}
\overline{\xi^2} = \overline{\eta^2} = \overline{\zeta^2} &= \frac{1}{12} \\
\overline{\xi^2\eta^2} = \overline{\eta^2\zeta^2} = \overline{\zeta^2\xi^2} &= \frac{1}{144}
\end{aligned} \tag{B.33}$$

Case 2:

Next assume that Ω is only the upper half of the box. Then $\xi_2 = \frac{1}{2}$, $\xi_1 = -\frac{1}{2}$, $\eta_2 = \frac{1}{2}$, $\eta_1 = -\frac{1}{2}$, $\zeta_2 = \frac{1}{2}$, $\zeta_1 = 0$, If L is even or M is even the corresponding mean value

vanishes. However the same thing is not true for N. The non-zero mean values are now

$$\begin{aligned} \overline{\zeta} &= \frac{1}{4} \\ \overline{\xi^2} = \overline{\eta^2} = \overline{\zeta^2} &= \frac{1}{12} \\ \overline{\zeta\xi^2} = \overline{\zeta\eta^2} &= \frac{1}{48} \\ \overline{\zeta^2\xi^2} = \overline{\zeta^2\eta^2} = \overline{\xi^2\eta^2} &= \frac{1}{144} \end{aligned} \tag{B.34}$$

Let us go back to Case 1 above and assume $\Delta x = \Delta y = \Delta z = 1$. Substituting Eqn. (B.33) into (B.27) and (B.29) we get

$$\vec{a}_\infty = (0, U_\infty, V_\infty, W_\infty, 0, 0, 0, 0) \tag{B.35}$$

and

$$C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{48} \end{pmatrix} \tag{B.36}$$

Hence,

$$\delta J = \rho_o[\Delta J_1 + \Delta J_2 + \Delta J_3 + \Delta J_4 + \Delta J_5 + \Delta J_6 + \Delta J_7 + \Delta J_8] \tag{B.37}$$

where

$$\begin{aligned} \Delta J_1 &= \delta\phi_1(+\frac{U_\infty}{4} + \frac{V_\infty}{4} + \frac{W_\infty}{4} - \frac{\phi_1}{3} + \frac{\phi_4}{12} + \frac{\phi_6}{12} + \frac{\phi_7}{12} + \frac{\phi_8}{12}) \\ \Delta J_2 &= \delta\phi_2(-\frac{U_\infty}{4} + \frac{V_\infty}{4} + \frac{W_\infty}{4} - \frac{\phi_2}{3} + \frac{\phi_3}{12} + \frac{\phi_5}{12} + \frac{\phi_7}{12} + \frac{\phi_8}{12}) \\ \Delta J_3 &= \delta\phi_3(+\frac{U_\infty}{4} - \frac{V_\infty}{4} + \frac{W_\infty}{4} + \frac{\phi_2}{12} - \frac{\phi_3}{3} + \frac{\phi_5}{12} + \frac{\phi_6}{12} + \frac{\phi_8}{12}) \\ \Delta J_4 &= \delta\phi_4(-\frac{U_\infty}{4} + \frac{V_\infty}{4} - \frac{W_\infty}{4} + \frac{\phi_1}{12} - \frac{\phi_4}{3} + \frac{\phi_5}{12} + \frac{\phi_6}{12} + \frac{\phi_7}{12}) \end{aligned}$$

$$\Delta J_5 = \delta\phi_5(+\frac{U_\infty}{4} + \frac{V_\infty}{4} - \frac{W_\infty}{4} + \frac{\phi_2}{12} + \frac{\phi_3}{12} + \frac{\phi_4}{12} - \frac{\phi_5}{3} + \frac{\phi_8}{12})$$
$$\Delta J_6 = \delta\phi_6(-\frac{U_\infty}{4} + \frac{V_\infty}{4} - \frac{W_\infty}{4} + \frac{\phi_1}{12} + \frac{\phi_3}{12} + \frac{\phi_4}{12} - \frac{\phi_6}{3} + \frac{\phi_7}{12})$$
$$\Delta J_7 = \delta\phi_7(+\frac{U_\infty}{4} - \frac{V_\infty}{4} - \frac{W_\infty}{4} + \frac{\phi_1}{12} + \frac{\phi_2}{12} + \frac{\phi_4}{12} + \frac{\phi_6}{12} - \frac{\phi_7}{3})$$
$$\Delta J_8 = \delta\phi_8(-\frac{U_\infty}{4} - \frac{V_\infty}{4} - \frac{W_\infty}{4} + \frac{\phi_1}{12} + \frac{\phi_2}{12} + \frac{\phi_3}{12} + \frac{\phi_5}{12} - \frac{\phi_8}{3})$$

The quantities inside the parentheses are the contribution of the box to the operators at each of the eight corner points respectively. For a uniform grid, each corner point of the grid gets contributions to its operator from each of the eight surrounding boxes. Consider the center grid point in Figure B.2. For the box in the upper right corner this point is local corner point number 1 and gets the contribution in Eqn. (B.37) from the coefficient of $\delta\phi_1$. However for the box in the lower left corner this point is local corner point number 8 and gets the contribution from the coefficient of $\delta\phi_8$. Note that if ρ_o is different in each of the eight boxes then we must add the terms in parentheses weighted by each ρ_o. However let us assume $M_\infty = 0$ in which case $\rho_o = 1$. Then when we add up all eight box contributions to the center grid point we get the coefficients shown in Figure B.2.

Note that the terms in $U_\infty, V_\infty, W_\infty$ all cancel out. This is due to the fact that $\nabla \cdot \vec{V}_\infty = 0$. The operator coefficients in Figure B.2 are known as the Bateman Laplacian, which is a second-order accurate approximation to $\nabla^2\phi$. This is not the same as the more diagonally dominant finite difference Laplacian (which we call a "lumped" Laplacian) shown in Figure B.3. The 7-point Laplacian can be obtained from a finite element point of view in a variety of ways. One way is to add higher order terms to the trilinear basis function; see Section 2.3. Another way is to add higher order terms to the Bateman principle itself. In the latter method one adds to δJ the term

$$\begin{aligned}\delta J \approx -\rho_o \int_\Omega \; [\; & \frac{1}{6}\{\frac{\Delta x^2}{2}({V_x}^2 + {W_x}^2) + \frac{\Delta y^2}{2}({W_y}^2 + {U_y}^2) \\ & + \frac{\Delta z^2}{2}({U_z}^2 + {V_z}^2)\} \\ & + \frac{1}{36}\{\frac{\Delta y^2 \Delta z^2}{2}{U_{yz}}^2 + \frac{\Delta z^2 \Delta x^2}{2}{V_{zx}}^2 \\ & + \frac{\Delta x^2 \Delta y^2}{2}{W_{xy}}^2\}]d\Omega \end{aligned} \qquad \text{(B.38)}$$

These "lumping" terms are negligible to second-order so that they do not affect the accuracy of the resultant operator.

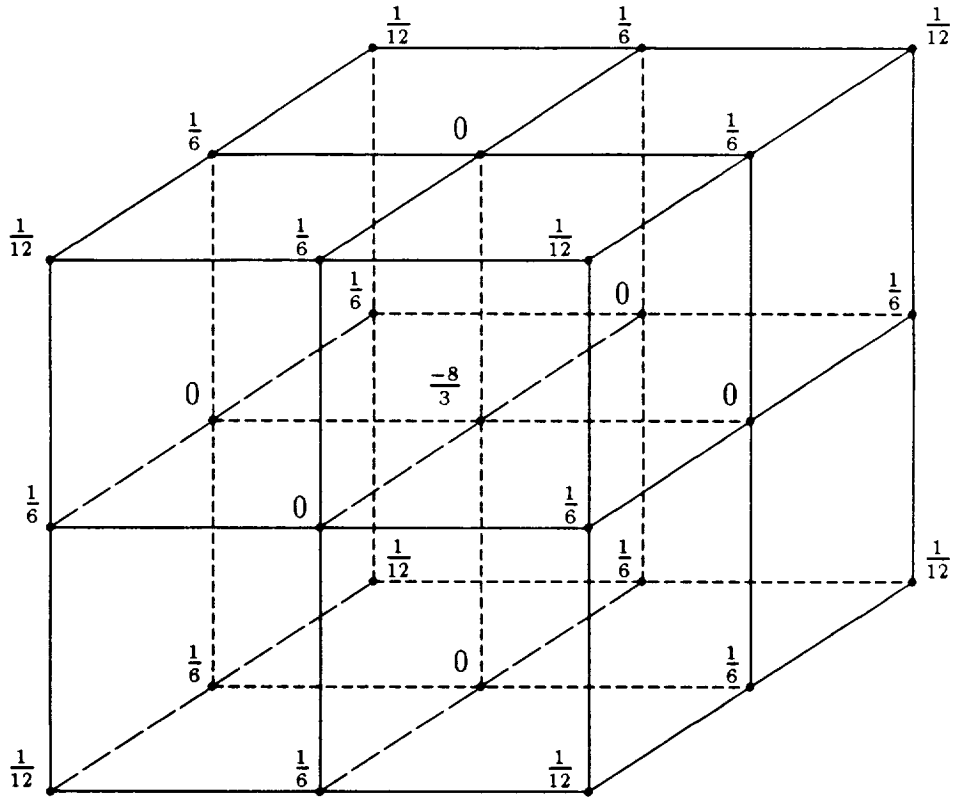


Figure B.2: Bateman Laplace Coefficients

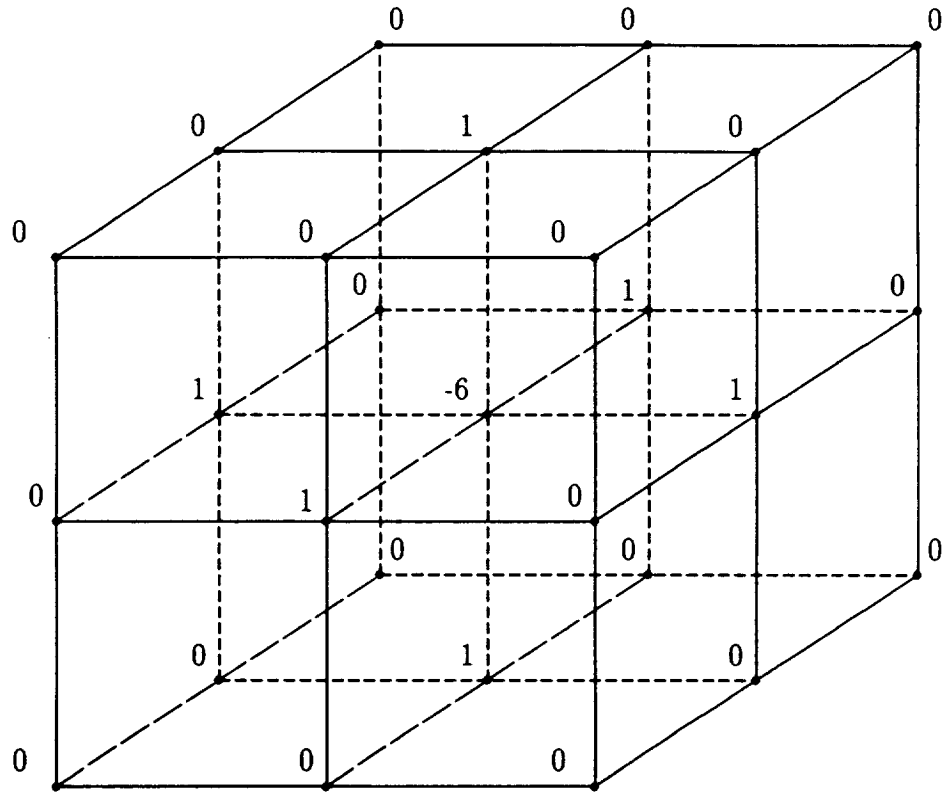


Figure B.3: Standard 7-Point Laplacian

We leave it to the reader to show that the lumped formula analogous to Eqn. (B.37) is

$$\delta J = \rho_o[\Delta J_1 + \Delta J_2 + \Delta J_3 + \Delta J_4 + \Delta J_5 + \Delta J_6 + \Delta J_7 + \Delta J_8] \tag{B.39}$$

where

$$\begin{aligned}
\Delta J_1 &= \delta\phi_1(+\frac{1}{4}U_\infty + \frac{1}{4}V_\infty + \frac{1}{4}W_\infty - \frac{3}{4}\phi_1 + \frac{1}{4}\phi_2 + \frac{1}{4}\phi_3 + \frac{1}{4}\phi_5)\\
\Delta J_2 &= \delta\phi_2(-\frac{1}{4}U_\infty + \frac{1}{4}V_\infty + \frac{1}{4}W_\infty - \frac{1}{4}\phi_1 - \frac{3}{4}\phi_2 + \frac{1}{4}\phi_4 + \frac{1}{4}\phi_6)\\
\Delta J_3 &= \delta\phi_3(+\frac{1}{4}U_\infty - \frac{1}{4}V_\infty + \frac{1}{4}W_\infty + \frac{1}{4}\phi_1 - \frac{3}{4}\phi_3 + \frac{1}{4}\phi_4 + \frac{1}{4}\phi_7)\\
\Delta J_4 &= \delta\phi_4(-\frac{1}{4}U_\infty - \frac{1}{4}V_\infty + \frac{1}{4}W_\infty + \frac{1}{4}\phi_2 + \frac{1}{4}\phi_3 - \frac{3}{4}\phi_4 + \frac{1}{4}\phi_8)\\
\Delta J_5 &= \delta\phi_5(+\frac{1}{4}U_\infty + \frac{1}{4}V_\infty - \frac{1}{4}W_\infty + \frac{1}{4}\phi_1 - \frac{3}{4}\phi_5 + \frac{1}{4}\phi_6 + \frac{1}{4}\phi_7)\\
\Delta J_6 &= \delta\phi_6(-\frac{1}{4}U_\infty + \frac{1}{4}V_\infty - \frac{1}{4}W_\infty + \frac{1}{4}\phi_2 + \frac{1}{4}\phi_5 - \frac{3}{4}\phi_6 + \frac{1}{4}\phi_8)\\
\Delta J_7 &= \delta\phi_7(+\frac{1}{4}U_\infty - \frac{1}{4}V_\infty - \frac{1}{4}W_\infty + \frac{1}{4}\phi_3 + \frac{1}{4}\phi_5 - \frac{3}{4}\phi_7 + \frac{1}{4}\phi_8)\\
\Delta J_8 &= \delta\phi_8(-\frac{1}{4}U_\infty - \frac{1}{4}V_\infty - \frac{1}{4}W_\infty + \frac{1}{4}\phi_4 + \frac{1}{4}\phi_6 + \frac{1}{4}\phi_7 + \frac{3}{4}\phi_8)
\end{aligned}$$

Note that one can derive the coefficients in Figure B.3 by adding the appropriate contributions from each of the eight surrounding boxes using Eqn. (B.39).

B.2 TEST CASE

Now let us consider a test case, i.e., flow past a one panel thin wing at an angle-of-attack. For simplicity a uniformly refined grid is chosen. The geometry is shown in Figure B.4. The grid has the dimensions $NX = 7, NY = 4, NZ = 4$, and the $y = 0$ plane is considered to be a plane of symmetry. The grid points are numbered in increasing x, then y, then z order. The wing has a chord of 2.5 and a span of 3. A wake panel is attached to the trailing edge. The wing plane is $z = 0$. In the bottom part of Figure B.4 we show the wing planform embedded in the grid, but actually there are no grid points at $z = 0$, and the points shown should be considered as projections.

Any grid box containing any part of the boundary surface is called a T-box. There are 10 T-boxes in the example. Any connected portion of a T-box is called a D-region. There are 14 D-regions in this example. The D-region numbers are shown in Figure B.4. The ordering of the D-regions is somewhat arbitrary and depends on which T-box is processed first and also on the panel normal direction. (In this case the wing upper normal is in the $+z$ direction and the wake upper normal is in the $-z$ direction.) Some T-boxes have one D-region and others have two. For example the inboard T-box at the wing leading edge has only one D-region (D-region 1) because the wing does not fully cut the T-box in half. Consider D-region 4 as shown in detail in Figure B.5. This D-region has a trilinear basis function defined by Eqn. (B.14). However, the values of ϕ at the corner points below the wing cannot be used in Eqn. (B.13) because the wing introduces a discontinuity in ϕ. Instead the values of ϕ below the wing are replaced by extrapolated values from above the wing. These values are denoted by ψ. We introduce an extrapolated ψ any time a ϕ is cut off from a neighboring ϕ in all four T-boxes adjoining the line segment connecting the two grid points. Note that we do not introduce a ψ simply because the two grid points are separated in some T-box. In the thin wing problem of Figure B.4, eight ψ's are introduced as shown. All lie on the plane of symmetry. The original problem contained unknowns ϕ_1 through ϕ_{112} ordered in grid order. The ψ's are considered as additional unknown ϕ's, i.e., $\psi_1 = \phi_{113}$ and $\psi_2 = \phi_{114}$, etc. The corner unknowns for D-region 4 are shown in Figure B.5. Now we proceed to derive operators for ϕ_{60} and ψ_1. First we consider the contribution to these operators from D-region 4, i.e., a formula analogous to Eqn. (B.37). (Remember $\phi_1, \phi_2, ...\phi_8$ are the local box values of ϕ here and should not be confused with the global numbering for the problem.)

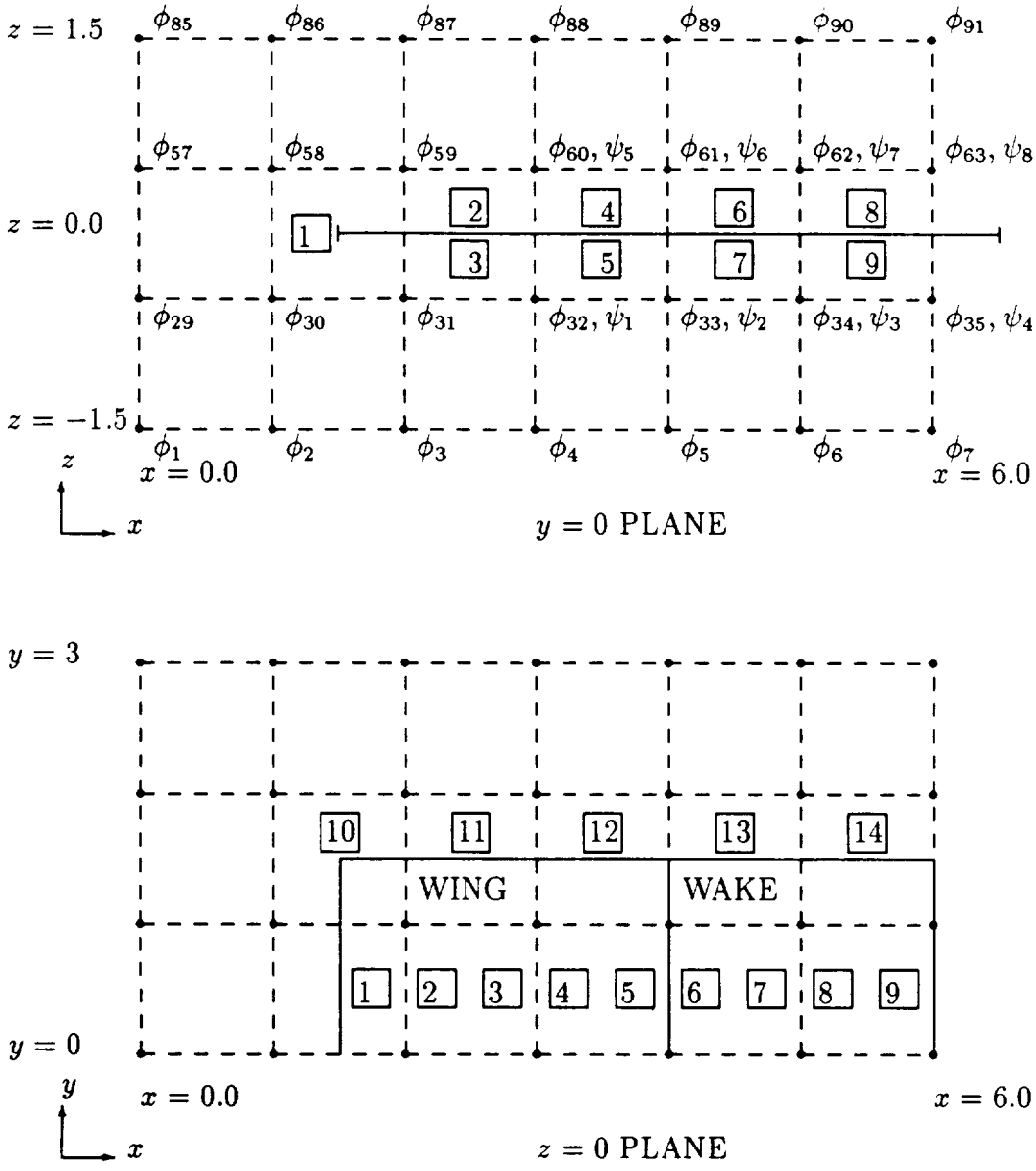


Figure B.4: One Panel Wing

One can use Eqn. (B.34) in Eqn. (B.29) and Eqn. (B.27) to obtain

$$\delta J = \rho_o[\delta J_1 + \delta J_2 + \delta J_3 + \delta J_4 + \delta J_5 + \delta J_6 + \delta J_7 + \delta J_8] \tag{B.40}$$

where

$$\begin{aligned}
\delta J_1 &= \delta\phi_1(+\frac{U_\infty}{16} + \frac{V_\infty}{16} + \frac{W_\infty}{8} - \frac{\phi_1}{12} - \frac{\phi_2}{48} - \frac{\phi_3}{48} + \frac{\phi_6}{24} + \frac{\phi_7}{24} + \frac{\phi_8}{24}) \\
\delta J_2 &= \delta\phi_2(-\frac{U_\infty}{16} + \frac{V_\infty}{16} + \frac{W_\infty}{8} - \frac{\phi_1}{48} - \frac{\phi_2}{12} - \frac{\phi_4}{48} + \frac{\phi_5}{24} + \frac{\phi_7}{24} + \frac{\phi_8}{24}) \\
\delta J_3 &= \delta\phi_3(+\frac{U_\infty}{16} - \frac{V_\infty}{16} + \frac{W_\infty}{8} - \frac{\phi_1}{48} - \frac{\phi_3}{12} - \frac{\phi_4}{48} + \frac{\phi_5}{24} + \frac{\phi_6}{24} + \frac{\phi_8}{24}) \\
\delta J_4 &= \delta\phi_4(-\frac{U_\infty}{16} - \frac{V_\infty}{16} + \frac{W_\infty}{8} - \frac{\phi_2}{48} - \frac{\phi_3}{48} - \frac{\phi_4}{12} + \frac{\phi_5}{24} + \frac{\phi_6}{24} + \frac{\phi_7}{24}) \\
\delta J_5 &= \delta\phi_5(+\frac{U_\infty}{16} + \frac{V_\infty}{16} - \frac{W_\infty}{8} + \frac{\phi_2}{24} + \frac{\phi_3}{24} + \frac{\phi_4}{24} - \frac{\phi_5}{4} + \frac{\phi_6}{48} + \frac{\phi_7}{48} + \frac{\phi_8}{12}) \\
\delta J_6 &= \delta\phi_6(-\frac{U_\infty}{16} + \frac{V_\infty}{16} - \frac{W_\infty}{8} + \frac{\phi_1}{24} + \frac{\phi_3}{24} + \frac{\phi_4}{24} + \frac{\phi_5}{48} - \frac{\phi_6}{4} + \frac{\phi_7}{12} + \frac{\phi_8}{48}) \\
\delta J_7 &= \delta\phi_7(+\frac{U_\infty}{16} - \frac{V_\infty}{16} - \frac{W_\infty}{8} + \frac{\phi_1}{24} + \frac{\phi_2}{24} + \frac{\phi_4}{24} + \frac{\phi_5}{48} + \frac{\phi_6}{12} - \frac{\phi_7}{4} + \frac{\phi_8}{48}) \\
\delta J_8 &= \delta\phi_8(-\frac{U_\infty}{16} - \frac{V_\infty}{16} - \frac{W_\infty}{8} + \frac{\phi_1}{24} + \frac{\phi_2}{24} + \frac{\phi_3}{24} + \frac{\phi_5}{12} + \frac{\phi_6}{48} + \frac{\phi_7}{48} - \frac{\phi_8}{4})
\end{aligned}$$

(Note that Ω, the volume of the D-region is $\frac{1}{2}$ in this case.)

Now ψ_1 is affected by D-regions 2 and 4. Because of the plane of symmetry one should also add in the contributions of the reflection of D-regions 2 and 4 across the $y = 0$ plane. For this purpose we simply reflect Figure B.5, i.e., in this case, interchange ψ_1 and ϕ_{39}, ψ_2 and ϕ_{40}, ϕ_{60} and ϕ_{67}, and ϕ_{61} and ϕ_{68}. Thus for the contribution of the unreflected D-region 4 to ψ_1, we choose the contribution due to $\delta\phi_1$ in Eqn. (B.40). However, for the contribution of the reflected D-region we choose the contribution due to $\delta\phi_3$. Summing up all contributions, we get the operator shown in Figure B.6. For comparison the operators generated in the code are also tabulated. This printout shows the coefficients of the ψ_1 operator (INDOP=1) located at grid point 32 (LOCOP=32) due to contributions from 2 physical boxes (NPRINT=2). The freestream coefficients are the coefficients of U_∞, V_∞ and W_∞, respectively. The remaining coefficients correspond to the unknowns indexed on the left, respectively. (The comparison between these formulas and those of the code are not exact, as the code shifts the wing in the grid by a slight amount in order to avoid panel surfaces landing right on grid lines. This shift also makes the ψ_1 operator formula depend minutely on some other unknowns due to the fact that a small amount of the wake is shifted into D-regions 4 and 5. These minute quantities can be ignored).

The ψ_1 operator has an interesting interpretation. Writing out the operator equations from Figure B.6 we have

$$-\frac{1}{24}\phi_{31} - \frac{1}{3}\psi_1 - \frac{1}{24}\psi_2 - \frac{1}{12}\phi_{39} \quad + \quad \frac{1}{12}\phi_{59}$$

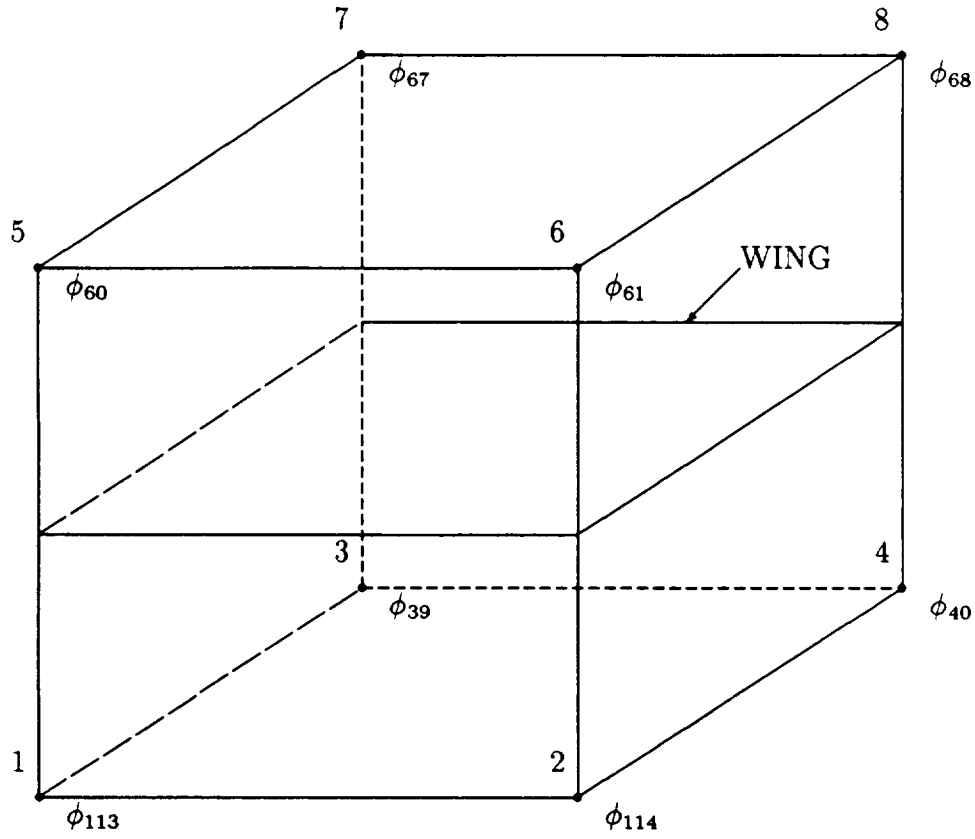


Figure B.5: D-region 4

$$+\frac{1}{12}\phi_{61}+\frac{1}{12}\phi_{66}+\frac{1}{6}\phi_{67}+\frac{1}{12}\phi_{68} \quad + \quad \frac{1}{2}W_\infty = 0 \tag{B.41}$$

This equation can be rewritten in a more suggestive way as

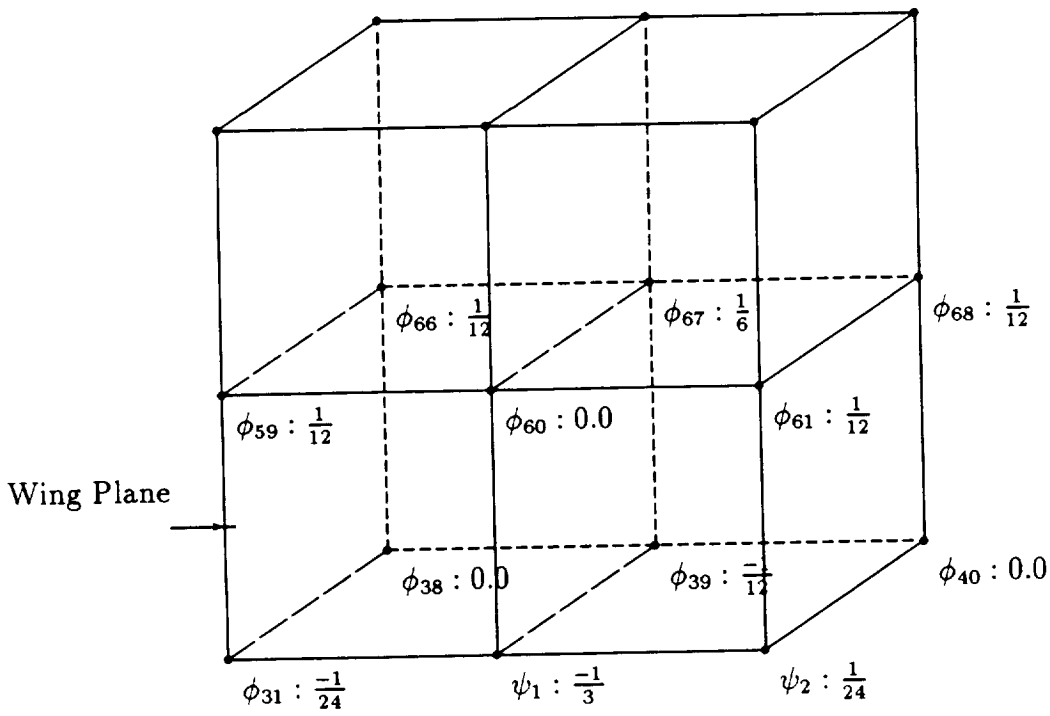
$$\begin{aligned}\frac{\Delta x\Delta y\Delta z}{2}[\frac{1}{\Delta z}(D_z{}^+\psi_1 + W_\infty) - \quad & \tfrac{1}{12} \quad (D_x{}^2\psi_1 + D_y{}^2\psi_1) \\ +\frac{1}{3}(D_x{}^2\phi_{60} + D_y{}^2\phi_{60}) + \quad & \tfrac{1}{12} \quad \Delta x\Delta y D_x{}^2 D_y{}^2\phi_{60}] = 0\end{aligned} \tag{B.42}$$

where

$$\begin{aligned} D_z{}^+\psi_1 &= \frac{\phi_{60}-\psi_1}{\Delta z} \\ D_x{}^2\psi_1 &= \frac{\psi_2 - 2\psi_1 + \phi_{31}}{\Delta x^2} \\ D_y{}^2\psi_1 &= \frac{\phi_{39} - 2\psi_1 + \phi_{39}}{\Delta y^2} \end{aligned}$$

etc...

Perturbation Coefficients



Free Stream Coefficients

U_∞	0.0	V_∞	0.0	W_∞	0.5

ϕ, ψ Coefficients

31	-0.41667E-01	32	-0.23582E-20	33	-0.22179E-15
38	0.53951E-06	39	-0.83334E-01	40	0.53954E-06
59	0.83335E-01	60	-0.84974E-06	61	0.83335E-01
66	0.83336E-01	67	0.16667E-00	68	0.83335E-01
113	-0.33334E-01	114	-0.41667E-01	117	-0.45336E-15
118	-0.42667E-10	121	0.49778E-10	122	0.14222E-10

Figure B.6: Operator Coefficients for $\psi_1 = \phi_{113}$

This is the way the equation would have looked had Δx , Δy , Δz been arbitrary. Note that in the limit as $\Delta z \to 0$ the equation tends to $D_z{}^+\psi_1 + W_\infty = 0$, which implies that the z velocity on the wing is zero, i.e., the wing is an impermeable surface. Thus the ψ_1 equation effectively imposes the appropriate boundary condition. To get the operator for ϕ_{60} we note that it has contributions from D-regions 2 and 4 as well as the free space boxes 59 and 60 (all boxes are numbered according to the index of the lower left corner point.) For free space boxes (i.e., boxes not containing a boundary) we always use the lumped formula (B.39) so that away from the boundary we get the 7-point Laplacian, consistent with the discrete Green's function. For D-regions 2 and 4 we choose the appropriate contribution in Eqn. (B.40) and for boxes 59 and 60 we choose the appropriate contribution from Eqn. (B.39). We must also include the contributions from the reflections of all four boxes, as they lie on a symmetry plane. The operator coefficients for ϕ_{60} are displayed in Figure B.7 and compare with those from TRANAIR shown also in the same figure.

For operators getting contributions from D-regions 6, 7, 8, 9, 13 and 14 the computation gets more complicated because the boundaries involve a wake. Specifically we have to add a surface integral to Eqn. (B.5) as in Eqn. (2.17) which is rewritten in the form

$$J = J + \int_\Sigma \alpha[\hat{n}\cdot \vec{W}](\Delta\Phi - \mu)\, d\Sigma \tag{B.43}$$

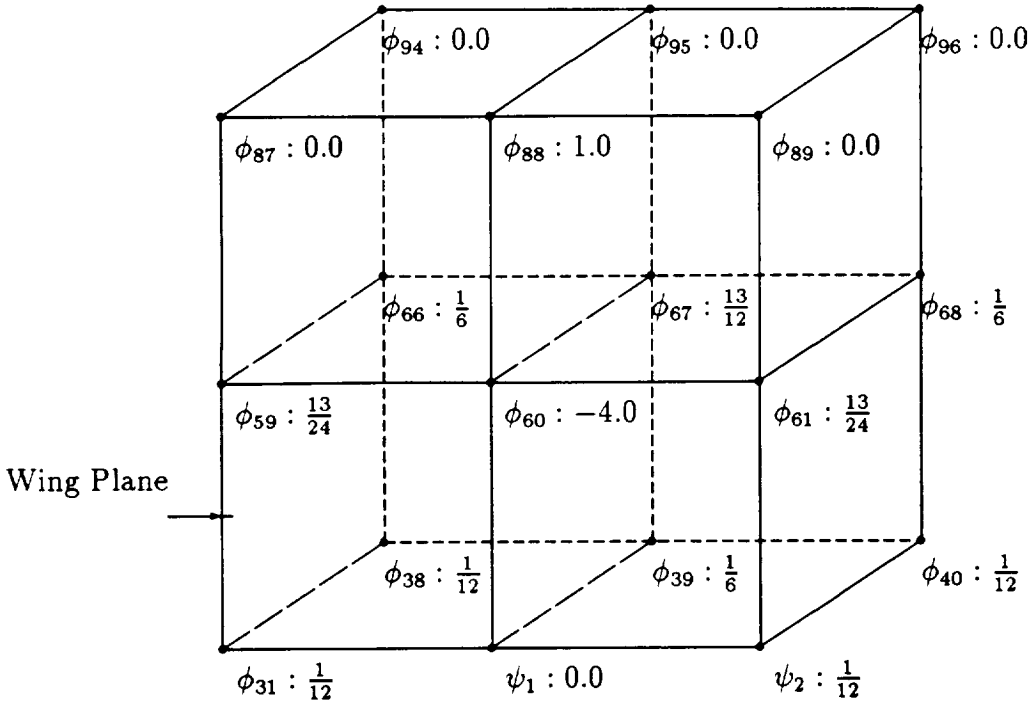
Taking a variation of the surface integral Eqn. (2.17) with respect to ϕ and assuming incompressible flow we get

$$\begin{aligned}\delta J = \rho_\circ \int_{\text{wake}} \; [\;& (\delta\phi_L - \delta\phi_U)(\vec{V}_\infty \cdot \hat{n}) \\ + \;& \frac{1}{2}(\delta\vec{V}_U \cdot \hat{n} + \delta\vec{V}_L \cdot \hat{n})(\mu + \phi_L - \phi_U) \\ + \;& \frac{1}{2}(\vec{V}_U \cdot \hat{n} + \vec{V}_L \cdot \hat{n})(\delta\phi_L - \delta\phi_U)]\, d\Sigma\end{aligned} \tag{B.44}$$

Here $\hat{n}$ is the upper normal on the wake. (In this case $\hat{n}$ points in the $-z$ direction.) $\vec{V}_U$ is $\nabla\phi$ evaluated using the basis function on the upper side of the wake and $\vec{V}_L$ is $\nabla\phi$ evaluated using the basis function on the lower side of the wake. Note that upper and lower mean the same thing as in PAN AIR. In this particular case they do not physically correspond to upper and lower surfaces. The surface integral in Eqn. (B.44) further requires the definition of how μ varies in the wake. We assume μ varies linearly from corner point to corner point. We leave it to the reader to determine the contributions to the operators due to Eqn. (B.44).

Note that the ψ_3 operator involves unknowns on the other side of the wake. This is clear physically since the wake is a jump discontinuity surface rather than a solid surface. It is also clear mathematically from Eqn. (B.44) where terms involving products of upper and lower basis functions are present. The operators shown in Figures B.6 and B.6 involve the unknowns $\mu_1 = \phi_{121}$ and $\mu_2 = \phi_{122}$, the values of doublet strength at the leading edge corner points of the wake network. μ_1 and μ_2 have their own operators, namely μ_1 should be the difference between the basis

Perturbation Coefficients



Lower Boxes Un-lumped. Upper Boxes Lumped.

Free Stream Coefficients

U_∞	0.0	V_∞	0.0	W_∞	0.5

ϕ, ψ Coefficients

31	0.83335E-01	32	-0.45366E-13	33	0.42667E-10
38	0.83335E-01	39	0.16667E-00	40	0.83335E-01
59	0.54167E-00	60	-0.40000E+01	61	0.54167E-01
66	0.16667E-00	67	0.10833E+01	68	0.16667E-00
87	0.00000E-00	88	0.10000E+01	89	0.00000E-00
94	0.00000E-00	95	0.00000E+00	96	0.00000E-00
113	-0.84974E-06	114	0.83335E-01	117	0.23581E-20
118	0.22178E-15	121	-0.49778E-10	122	-0.14222E-10

Figure B.7: Operator Coefficients for ϕ_{60}

function of D- region 6 and that of D-region 7 evaluated at the inboard wake leading edge corner point. Similarly, μ_2 is the difference between the basis function of D-region 13 and that of D-region 13 at the outboard wake leading edge corner point. (Since there is no difference, $\mu_2 = 0$ as it should be.)

The operator coefficients are displayed visually in Figures B.8 and B.9, and they have an interesting interpretation.

In the same way as in Eqn. (B.42) the ψ_3 and ψ_7 operators can be rewritten suggestively in the form

$$\begin{aligned}\frac{\Delta x\Delta y\Delta z}{2} \quad [\quad & \frac{1}{\Delta z^2}(\phi_{62} - \psi_7 + \mu_1) + \frac{1}{3}D_y{}^+\mu_1 \\ + \quad & \frac{1}{12}D_x{}^2\psi_3 + \frac{7}{18}D_x{}^2\phi_{62} - \frac{1}{9}D_x{}^2\psi_7 \\ + \quad & \frac{1}{36}D_y{}^2\psi_2 + \frac{1}{36}D_y{}^2\psi_3 + \frac{1}{36}D_y{}^2\psi_4 \\ + \quad & \frac{1}{9}D_y{}^2\phi_{61} + \frac{1}{9}D_y{}^2\phi_{62} + \frac{1}{9}D_y{}^2\phi_{63} \\ - \quad & \frac{1}{18}D_y{}^2\psi_6 - \frac{1}{18}D_y{}^2\psi_7 - \frac{1}{18}D_y{}^2\psi_8] = 0 \end{aligned} \tag{B.45}$$

$$\begin{aligned}\frac{\Delta x\Delta y\Delta z}{2} \quad [\quad & \frac{1}{\Delta z^2}(\phi_{34} - \psi_3 - \mu_1) - \frac{1}{3}D_y{}^+\mu_1 \\ + \quad & \frac{1}{12}D_x{}^2\psi_7 + \frac{7}{18}D_x{}^2\phi_{34} - \frac{1}{9}D_x{}^2\psi_4 \\ + \quad & \frac{1}{36}D_y{}^2\psi_6 + \frac{1}{36}D_y{}^2\psi_7 + \frac{1}{36}D_y{}^2\psi_8 \\ + \quad & \frac{1}{9}D_y{}^2\phi_{33} + \frac{1}{9}D_y{}^2\phi_{34} + \frac{1}{9}D_y{}^2\phi_{35} \\ - \quad & \frac{1}{18}D_y{}^2\psi_2 - \frac{1}{18}D_y{}^2\psi_3 - \frac{1}{18}D_y{}^2\psi_4] = 0 \end{aligned} \tag{B.46}$$

Adding Eqn. (B.45) to (B.46) and ignoring higher order terms we get

$$\frac{\Delta x\Delta y\Delta z}{2} \cdot \frac{1}{\Delta z}[\frac{\phi_{62} - \psi_3}{\Delta z} - \frac{\psi_7 - \phi_{34}}{\Delta z}] = 0 \tag{B.47}$$

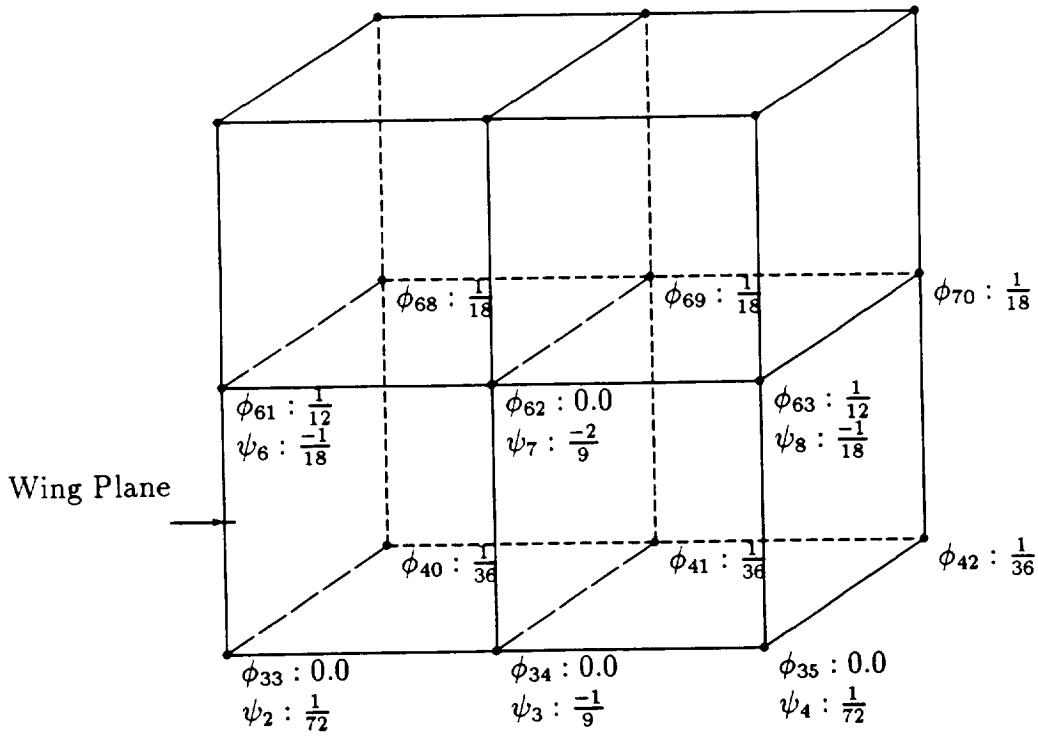
In the limit as $\Delta z \to 0$, this equation states that the z- velocity evaluated from the upper and lower basis functions is the same, i.e., the normal velocity is continuous across a wake.

Subtracting Eqn. (B.45) from Eqn. (B.46) and ignoring higher order terms we get

$$\frac{\Delta x\Delta y\Delta z}{2} \cdot \frac{2}{\Delta z^2}[\frac{\phi_{34} + \psi_7}{2} - \frac{\phi_{62} + \psi_3}{2} - \mu_1] = 0 \tag{B.48}$$

This equation states that the difference in potential between the two basis functions evaluated at the wake plane is equal to the wake doublet strength. We see then that the ψ equations give us the proper jump conditions across the wake.

Perturbation Coefficients



Free Stream Coefficients

U_∞	0.0	V_∞	0.0	W_∞	$\frac{1}{2}$

Doublet Coefficients

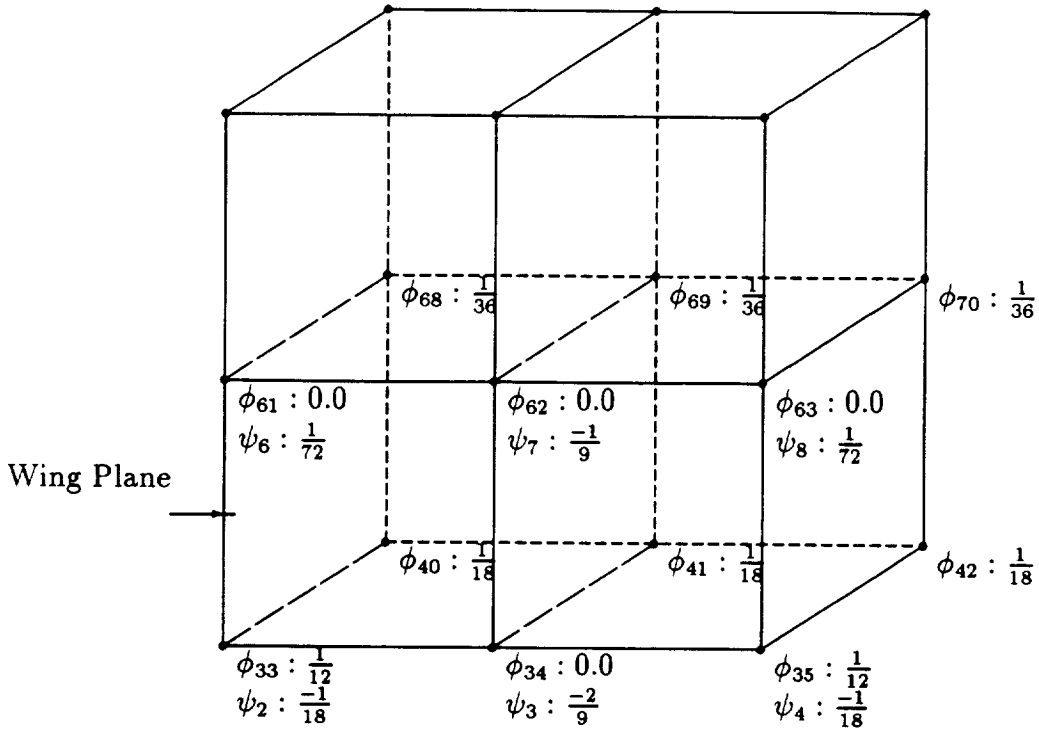
μ_1	$\frac{7}{18}$	μ_2	$\frac{1}{9}$		

ϕ, ψ Coefficients

33	-0.28878E-06	34	-0.11551E-05	35	-0.28878E-06
40	0.27779E-01	41	0.27779E-01	42	0.27779E-01
61	0.83334E-01	62	-0.20052E-05	63	0.83334E+01
68	0.55556E-01	69	0.55556E-01	70	0.55556E-01
114	0.13890E-01	115	-0.11111E-01	116	0.13890E+01
118	-0.55556E-01	119	-0.22222E+00	120	-0.55556E-01
121	0.38889E+00	122	0.11111E+00		

Figure B.8: Operator Coefficients for $\psi_3 = \phi_{115}$

Perturbation Coefficients



Free Stream Coefficients

U_∞	0.0	V_∞	0.0	W_∞	0.0

Doublet strength coefficients

μ_1	$\frac{-7}{18}$	μ_2	$\frac{-1}{9}$

ϕ, ψ Coefficients

33	0.83333E-01	34	0.26182E-05	35	0.83333E-01
40	0.55554E-01	41	0.55554E-01	42	0.55555E-01
61	0.28916E-06	62	0.11566E-05	63	0.28916E-06
68	0.27777E-01	69	0.27777E-01	70	0.27777E-01
114	-0.55555E-01	115	-0.22222E-00	116	-0.55555E-01
118	0.13888E-01	119	-0.11111E-00	120	0.13888E-01
121	-0.38889E-00	122	-0.11111E-00		

Figure B.9: Operator Coefficients for $\psi_7 = \phi_{119}$

As a final comment to this section we note that the lumping terms of Eqn. (B.38) can also be used in boxes which have D-regions to obtain a more diagonally dominant operator. However, for consistency all surface integrals in the D-region must also be lumped. As an example a surface term of the form

$$\delta J \approx -\rho_o \int_\Sigma (\hat{n} \cdot \vec{V}) \delta\phi d\Sigma \tag{B.49}$$

must be augmented by the term

$$\begin{aligned}
\delta J_{\text{extra}} = -\rho_o \int_\Sigma [\quad & n_x \quad \frac{1}{6}(\Delta y^2 U_y \delta\phi_y + \Delta z^2 U_z \delta\phi_z) \\
+ \quad & n_x \quad \frac{1}{36}(\Delta y^2 \Delta z^2 U_{yz} \delta\phi_{yz}) \\
+ \quad & n_y \quad \frac{1}{6}(\Delta z^2 V_z \delta\phi_z + \Delta x^2 V_x \delta\phi_x) \\
+ \quad & n_y \quad \frac{1}{36}(\Delta z^2 \Delta x^2 V_{zx} \delta\phi_{zx}) \\
+ \quad & n_z \quad \frac{1}{6}(\Delta x^2 W_x \delta\phi_x + \Delta y^2 W_y \delta\phi_y) \\
+ \quad & n_z \quad \frac{1}{36}(\Delta x^2 \Delta y^2 W_{xy} \delta\phi_{xy}) \quad] d\Sigma
\end{aligned} \tag{B.50}$$

B.3 PRESSURE BOUNDARY CONDITION

In this section we discuss implementation of the pressure boundary condition Eqn. (2.7). This boundary condition is generally imposed on wake sheets, which are wetted by the flow on both sides. Potential is allowed to jump across these sheets, and the value of this jump constitutes the degree of freedom which allows the imposition of Eqn. (2.7). Because TRANAIR has its roots in the panel method [8], we call the jump in potential across a wake 'doublet strength', and denote its value by μ.

In order to discretize Eqn. (2.7) we define doublet strength at various point locations on the wake sheets and then impose Eqn. (2.7) at a like number of discrete (but different locations). In Fig. B.10 we display a schematic of those wake surface discretizations which are operable in TRANAIR. These discretizations are comprised of networks of mesh points which are interpolated by the wake surfaces. The portion of the surfaces interpolating four adjacent mesh points is called a panel. In Fig. B.10 the lines correspond to panel edges and their intersections to mesh points. The mesh points are identified by a rectangular array of indices, (I=1,M) and (J=1,N), where I is the row index and J the column index. The upper surface of the network is defined to be that side whose normal corresponds (in a schematic sense) to the direction $\vec{N} \otimes \vec{M}$. It is useful to number the four network edges as shown in Fig. B.10. Discrete doublet parameters are located at positions denoted by the solid dot. The doublet strength on each panel is then obtained by bilinear interpolation. Pressure boundary conditions are imposed at the locations marked by $\times$'s. Edge 1 (the leading edge) is considered a special edge for each type of network. It is assumed that the mean (average of upper and lower) tangential velocity enters the network along this edge. Since the pressure jump condition will involve only derivatives of doublet strength, constants of integration must be directly or indirectly specified along the leading edge to fix the level of doublet strength along each streamline. Boundary conditions along the leading edge are called Kutta conditions and their discrete locations are denoted by the open dot.

Wake networks may abut other wake networks. In general, doublet strength must be continuous, so it is essential to ensure that the doublet parameters along the edge of one network match those along the edge of an adjoining network. Hence boundary conditions along edges of some networks may be replaced by explicit doublet matching conditions.

The network type designations (6,18 and 20) arise from historical considerations involving the panel method [8]. Network type 6 is a full wake network, where the only assumption required for its employment is that the mean flow enters the network at the leading edge. Network type 18 is a special case of network type 6 where the doublet strength at any corner point is assumed equal to that of the first point in the respective column. This means that the derivative of doublet strength along panel column edges is zero. In certain instances this implies that Eqn. (2.7) is satisfied automatically at locations similar to the $\times$'s for network type 6, which results in substantial savings. Generally one can employ type 18 networks in place of type 6 networks when the mean flow is roughly in the direction of panel column edges and the difference in total pressure and total temperature across the wake is not too great. Network type

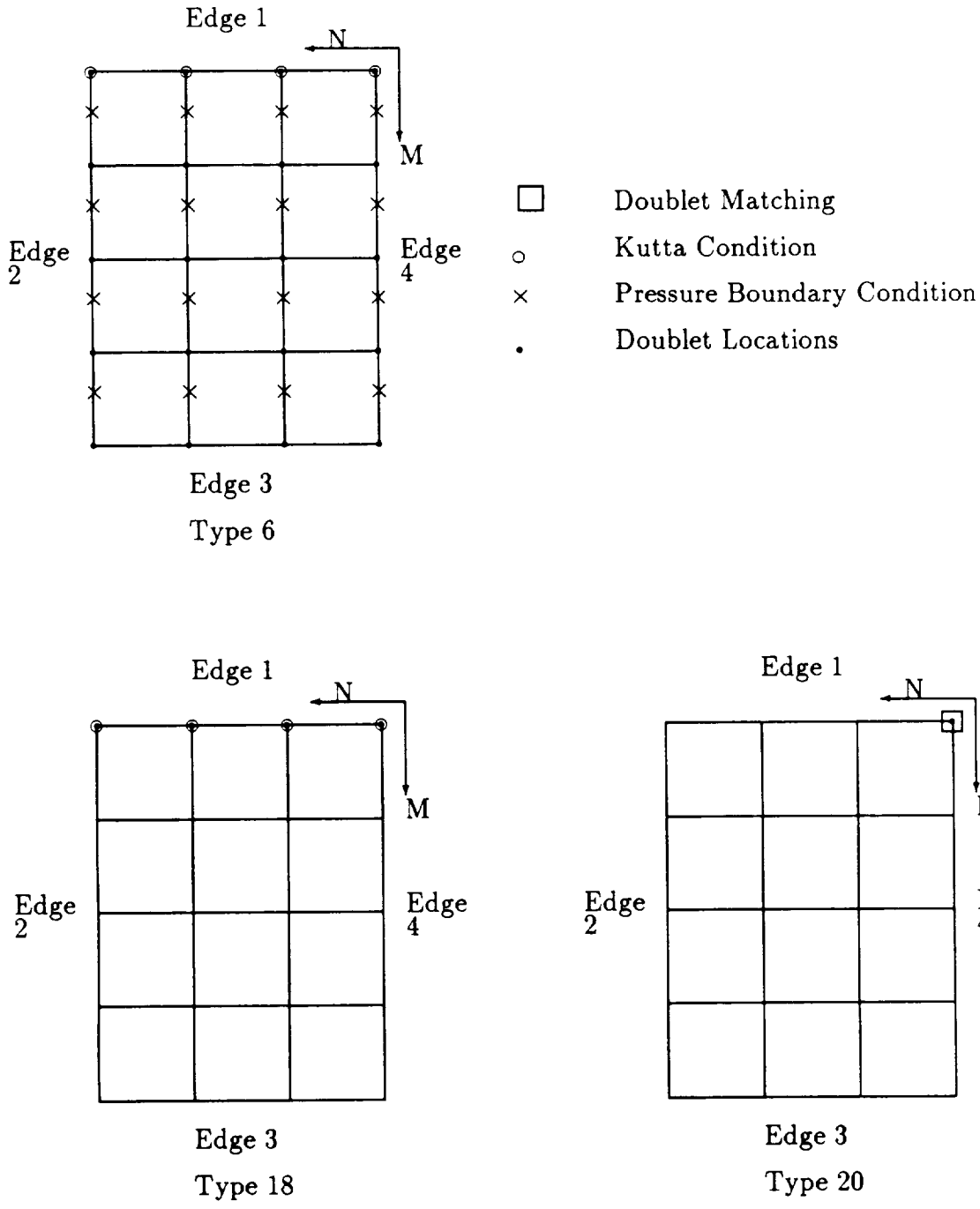


Figure B.10: Wake Networks

20 is a special case of network type 18 where the doublet strengths along the leading edge are all identical to that at the head of the first column. Hence it is a constant strength doublet network and contains no vorticity (or jump in tangential velocity) at all. Network type 20 is generally used as a circulation carry-over wake rather than to account for vortex separation. This is opposed to network types 6 and 18, where the Kutta condition at the leading edges is intended to force vortex separation along the line where these edges adjoin a solid body. (Note in this connection that the the sheet vorticity vector $\vec{\zeta}$ is equal to $\hat{n} \otimes \vec{\nabla}\mu$).

The imposition of Eqn. (2.7) at the points denoted by $\times$ in Fig. B.10 is relatively straightforward, but requires some care. For example, upper surface velocity $\vec{V}_u$ and lower surface velocity $\vec{V}_l$ may be evaluated by differentiating the potential basis functions on respective sides of the wakes. Then upper surface pressure p_u and lower surface pressure p_l may be calculated from Eqn. (2.14), and substituted into Eqn. (2.7). Formally Eqn. (2.7) does not involve μ , and μ is determined through its appearance in the finite element operators (e.g. see Eqn. (B.45)). This leads to conditioning problems since the number of doublet unknowns is unrelated to the number of finite element equations. It is preferable to get μ involved directly in Eqn. (2.7) through its definition as the jump in potential. For this purpose we redefine the upper and lower surface velocity vectors by the formulas:

$$\vec{V}_u' = \frac{1}{2}(\vec{V}_u + \vec{V}_l) + \frac{1}{2}\vec{\nabla}\mu - \left\{ \frac{\tilde{\vec{n}} \cdot \left[\vec{\nabla}\mu - (\vec{V}_u - \vec{V}_l)\right]}{\tilde{\vec{n}} \cdot \hat{n}} \right\} \hat{n} \tag{B.51}$$

$$\vec{V}_l' = \frac{1}{2}(\vec{V}_u + \vec{V}_l) - \frac{1}{2}\vec{\nabla}\mu - \left\{ \frac{\tilde{\vec{n}} \cdot \left[\vec{\nabla}\mu - (\vec{V}_u - \vec{V}_l)\right]}{\tilde{\vec{n}} \cdot \hat{n}} \right\} \hat{n} \tag{B.52}$$

Here, $\vec{V}_u$ and $\vec{V}_l$ are the upper and lower surface velocity vectors calculated by differentiating the respective potential basis functions, but the (mathematically) equivalent velocity vectors $\vec{V}_u'$ and $\vec{V}_l'$ are used to compute p_u and p_l. The quantity $\tilde{\vec{n}}$ is the surface co-normal vector defined by

$$\tilde{\vec{n}} = \begin{cases} ((1 - M_\infty{}^2)n_x, n_y, n_z) & \text{linear flow} \\ \hat{n} & \text{non-linear flow} \end{cases} \tag{B.53}$$

Note that

$$\vec{\zeta} = \hat{n} \otimes (\vec{V}_u' - \vec{V}_l') = \hat{n} \otimes \vec{\nabla}\mu, \tag{B.54}$$

i.e. the vorticity calculated from the velocities used in the pressure calculations is a function of μ only rather than dependent on the potential basis functions. In the case where there are no total pressure or temperature differences across the wake, Eqn. (2.7) implies

$$\left|\vec{V}_u'\right| = \left|\vec{V}_l'\right| \quad \text{and} \quad \rho_u = \rho_l. \tag{B.55}$$

Hence,

$$\alpha\vec{W} \cdot \Delta\vec{V}' = 0, \tag{B.56}$$

where $\alpha\vec{W} = \frac{1}{2}(\vec{W}_u + \vec{W}_l)$, $\Delta\vec{V}' = \vec{V}_u' - \vec{V}_l'$, and $\vec{W}_u = \rho_u\vec{V}_u$, $\vec{W}_l = \rho_l\vec{V}_l$.

Assuming the wake is a stream surface relative to the mean mass flux, i.e.

$$\hat{n} \cdot \alpha\vec{W} = 0 \tag{B.57}$$

we obtain the classical condition for determining wake doublet strength, i.e.,

$$\alpha\vec{W} \cdot \vec{\nabla}\mu = 0 \tag{B.58}$$

Notice that if the panel column edges are aligned along $\alpha\vec{W}$, then μ is constant along these edges and a type 18 network may be employed as mentioned above.

In the case where there are total pressure and temperature differences across the wake, then Eqn. (B.58) should be replaced by

$$\alpha\vec{W}^* \cdot \vec{\nabla}\mu^* = 0 \tag{B.59}$$

Here $\vec{W}^*$ is defined by Eqn. (2.22). Although Eqn. (B.59) is no longer exact, it is often a reasonably good approximation.

At Kutta condition locations we impose the following boundary condition:

$$\mu - (\phi_u - \phi_l) + \epsilon\hat{t} \cdot \left[\vec{\nabla}\mu - \vec{\nabla}(\phi_u - \phi_l)\right] = 0 \tag{B.60}$$

Here $\hat{t}$ is a unit tangent vector lying along the local panel column edge, and ϵ is a small parameter which is chosen to be approximately equal to the local field mesh size in the $\hat{t}$ direction. This equation guarantees that μ is the jump in basis function potential in the case where wake panels are denser than the field grid. The derivative term is added in the case where the previous condition is redundant with respect to the finite element operators. Then Eqn. (B.60) implies that the jump in basis function velocity is well defined and finite.

We conclude by noting that all the analysis contained in this section is applicable to the imposition of pressure boundary conditions on surfaces which are wetted by the fluid on one side only. Such a boundary condition is often used in the design mode, where pressure is specified and the surface is to be updated to be a stream surface in the resultant flow. To modify the analysis it is only necessary to replace the velocity $\vec{V}_l'$ by the velocity which would yield the desired upper surface pressure.

Appendix C

GMRES

In this appendix the algorithmic details of GMRES are described. GMRES algorithm is used as the iterative driver to to drive the residual to zero. Attention is given to the concept of preconditioning and the role it plays in assuring rapid convergence. The advantages GMRES enjoys over related methods such as conjugate gradients are explored.

C.1 GMRES ALGORITHM

GMRES [53] is a method for solving nonsymmetric linear systems of algebraic equations. A modified version of GMRES which applies to nonlinear systems of equations is described.

Consider a differentiable system

$$A(u) = 0 \tag{C.1}$$

of N nonlinear equations in N unknowns. The differential $\overline{A}(u;p)$ of A at u in the direction p is defined by

$$\overline{A}(u;p) = \lim_{\epsilon \to 0} \frac{A(u+\epsilon p) - A(u)}{\epsilon} \tag{C.2}$$

For computational purposes, ϵ is taken to be some small number and ensure that the variables u and the component values of $A(u)$ are reasonably scaled to permit an accurate evaluation of $\overline{A}(u;p)$.

Given u^n, an approximate solution to Eqn. (C.1), one cycle of GMRES advances the solution by first choosing k orthonormal search directions $p_1, p_2 ... p_k$ as follows:

$$p_1 = -A(u^n) \tag{C.3}$$

Normalize p_1

$$p_1 = \frac{p_1}{\| p_1 \|} \tag{C.4}$$

For $j = 1, 2, ...k-1$ take

$$p_{j+1} = \overline{A}(u^n; p_j) - \sum_{i=1}^{j} b_{ji} p_i \tag{C.5}$$

where

$$b_{ji} = (\overline{A}(u^n; p_j), p_i)$$

so that

$$(p_{j+1}, p_i) = 0 \quad \text{for} \quad \text{i} = 1, 2, ...\text{j}$$

Normalize p_{j+1}

$$p_{j+1} = \frac{p_{j+1}}{\| p_{j+1} \|} \tag{C.6}$$

Now update u^n using

$$u^{n+1} = u^n + \sum_{j=1}^{k} a_j p_j \tag{C.7}$$

The overall goal is to minimize $A(u^{n+1})$. To this end the coefficients a_j are chosen to solve the linearized version of the least squares problem

$$\begin{aligned} \| A(u^n) + \sum_{j=1}^{k} a_j \overline{A}(u^n; p_j) \|^2 &\simeq \| A(u^n + \sum_{j=1}^{k} a_j p_j) \|^2 \\ &= \| A(u^{n+1}) \|^2 \end{aligned} \tag{C.8}$$

Aided by the orthogonality of the search directions p_j, a modified version of the QR algorithm described in [53] is used to solve this least squares problem.

One cycle of the GMRES algorithm is an approximation to one cycle of Newton's iteration. Indeed with Newton's method one uses the linear approximation

$$A(u^{n+1} + p) \simeq A(u^n) + \overline{A}(u^n; p) \tag{C.9}$$

to estimate a value of p which will enable $A(u^n + p) = 0$. Eqn. (C.9) suggests that the following linear equation be solved for p:

$$A(u^n) + \overline{A}(u^n; p) = 0 \tag{C.10}$$

The naive way to do this is to compute the entries of the $N \times N$ matrix associated with $\overline{A}$ and directly solve the system of linear equations. This is enormously expensive if N is at all large (as it is for all problems of practical interest). GMRES approximately solves Eqn. (C.10) by finding the best possible solution over the k dimensional linear subspace spanned by the search directions $< p_1, p_2, ...p_k >$. Of course if $k = N$, then GMRES would find the best possible solution to Eqn. (C.10) over the whole space and would therefore compute the exact solution. Unfortunately this is every bit as expensive as solving Eqn. (C.10) directly. The key to efficiency is to arrange for GMRES to find a good solution to Eqn. (C.10) using only a small number of search directions k. Preconditioning plays a vital role in achieving this goal.

C.2 PRECONDITIONING

The available mathematical theorems [53],[88], as well as much numerical experience, indicate that the rate at which GMRES converges, measured by the value of k required to achieve a given level of accuracy, depends on the distribution of eigenvalues. The more the eigenvalues are clustered together, the faster GMRES will converge. The process of replacing a given problem with another equivalent problem (i.e., one with the same solution) enjoying a more favorable distribution of eigenvalues is called preconditioning.

Based on the observation that the identity operator has the most favorable distribution of eigenvalues (all the eigenvalues are clustered at 1), most methods for preconditioning invoke an approximate inverse to the operator in the equation one is solving. For example, if L is a linear operator and N^{-1} is an approximate inverse to L then the problem

$$L(x) - b = 0 \tag{C.11}$$

is equivalent to

$$N^{-1}(L(x) - b) = 0. \tag{C.12}$$

In this case N is called a preconditioner for L. These equations have the same solution, but GMRES will more readily solve Eqn. (C.12) than Eqn. (C.11) because the eigenvalues of $N^{-1}L$ are more tightly clustered than those of L.

A formulation which allows GMRES to take advantage of preconditioners already built into existing codes will now be described.
Given a problem of the form

$$A(u) = 0 \tag{C.13}$$

most computer codes have a method M which takes a good approximation to the solution of Eqn. (C.13) and creates a better approximation. Typical methods M might, for example, involve SLOR, ADI, a time marching scheme or even multigrid. Whatever the method, M already invokes an approximate inverse to the operator in Eqn. (C.13). The standard procedure for updating u is

$$u^{n+1} = M(u^n) \tag{C.14}$$

Convergence is achieved when $u^{n+1} = u^n$. Thus solving Eqn. (C.13) is equivalent to solving

$$u - M(u) = 0 \tag{C.15}$$

Applying GMRES to the preconditioned Eqn. (C.15) is more effective than applying GMRES to Eqn. (C.13) directly. Applying GMRES to Eqn. (C.15) is often considerably more effective than employing the standard iteration procedure Eqn. (C.14) [54].

C.3 GMRES AND SIMILAR METHODS

As shown above, one cycle of GMRES finds the best possible solution to the following linear equation for p:

$$A(u^n) + \overline{A}(u^n; p) = 0 \tag{C.16}$$

over a k dimensional subspace $< p_1, p_2, ...p_k >$. In GMRES the search directions are chosen to be orthonormal. An alternative method ORTHODIR, chooses the search directions to be AA^T orthogonal, i.e.,

$$(\overline{A}(u^n; p_i), \overline{A}(u^n; p_j)) = 0, \quad i \neq j \tag{C.17}$$

ORTHODIR computes the k search directions as:

$$p_1 = -A(u^n)$$

for $j = 1, 2, ...k-1$ take

$$v_j = \overline{A}(u^n; p_j)$$

$$p_{j+1} = v_j + \sum_{i=1}^{j} b_{ji} p_j$$

where

$$b_{ji} = \frac{(\overline{A}(u^n; v_j), \overline{A}(u^n; p_i))}{(\overline{A}(u^n; p_i), \overline{A}(u^n; p_i))}$$

The coefficients b_{ji} are computed to enforce Eqn. (C.17). As in GMRES, ORTHODIR updates u using the formula

$$u^{n+1} = u^n + \sum_{j=1}^{k} a_j p_j$$

where the a_j are chosen to solve the linearized least squares problem

$$\begin{aligned} \| A(u^n) + \sum_{j=1}^{k} a_j \overline{A}(u^n; p_j) \|^2 &\simeq \| A(u^n + \sum_{j=1}^{k} a_j p_j) \|^2 \\ &= \| A(u^{n+1}) \|^2 \end{aligned} \tag{C.18}$$

Now because of Eqn. (C.17) this least squares problem can be explicitly solved:

$$a_j = -\frac{(A(u^n), \overline{A}(u^n; p_j))}{(\overline{A}(u^n; p_j), \overline{A}(u^n; p_j))}$$

It can be easily shown that the search directions generated by ORTHODIR span the same k dimensional subspace as those generated by GMRES. Therefore GMRES

and ORTHODIR are mathematically equivalent. Since ORTHODIR solves the least squares problem more conveniently, ORTHODIR would appear to be preferable to GMRES. However, ORTHODIR requires that both the search directions p_j and the vectors $A(u^n; p_j)$ be stored. Also ORTHODIR uses $2k$ evaluations of A. GMRES on the other hand requires only $k+1$ evaluations of A and that only the p_j be stored. (The solution to the least squares problem Eqn. (C.8) presented in reference [53] makes use of Eqn. (C.5) which expresses $\overline{A}(u^n; p_j)$ as a linear combination of the search directions p_j, eliminating the need to explicitly store the vectors $A(u^n; p_j)$). Thus GMRES requires only about half the storage and half the number of function evaluations as ORTHODIR. Moreover, evidence is given in reference [53] that GMRES is less subject to numerical problems. Of all the methods for solving a nonsymmetric linear system of equations based on the idea of finding the best possible solution over a k dimensional subspace (ORTHOMIN, ORTHORES and ORTHODIR are compared in reference [53]), GMRES appears to be the best in terms of storage, operation count and numerical stability.

For problems involving symmetric positive definite matrices, algorithms more efficient than GMRES exist. Consider the linear system

$$Sx = b \tag{C.19}$$

where S is a symmetric positive definite matrix. The CR (Conjugate Residual) method employs the following relations [88]:

Choose: x_0

Set: $r_0 = b - Sx_0$

and $p_0 = r_0$

Now recursively use

$$\begin{aligned} a_i &= (r_i, Sr_i)/(Sp_i, Sp_i) \\ x_{i+1} &= x_i + a_i p_i \\ r_{i+1} &= r_i - a_i Sp_i \\ b_i &= \frac{(r_{i+1}, Sr_{i+1})}{(r_i, Sr_i)} \\ p_{i+1} &= r_{i+1} + b_i p_i \end{aligned}$$

It then follows [88] that

$$(Sp_i, Sp_j) = 0 \quad \text{for } i \neq j$$

and that x_k minimizes

$$\| Sx - b \|^2$$

over the k dimensional affine space $x_0 + < p_0, p_1, \cdots p_{k-1} >$.

Thus CR operates very much in the same spirit as GMRES (in fact for symmetric positive definite linear systems they are mathematically equivalent), but explicit orthogonalizations such as Eqn. (C.5) and the need for solving the least squares problem (C.8) are avoided. Moreover storage is required for only the 5 vectors $x, r, Sr, p,$ and Sp. This compares very favorably with the $k+4$ stored vectors required GMRES (typically 20 vectors).

Another algorithm for solving Eqn. (C.19) is Conjugate Gradients (CG). It uses the relations [88]:
Choose x_0. Set

$$r_0 = b - Sx_0 \tag{C.20}$$

and

$$p_0 = r_0 \tag{C.21}$$

Now recursively use

$$\begin{aligned} a_i &= \frac{(r_i, r_i)}{(p_i, Sp_i)} \\ x_{i+1} &= x_i + a_i p_i \\ r_{i+1} &= r_i - a_i S p_i \\ b_i &= \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)} \\ p_{i+1} &= r_{i+1} - b_i p_i \end{aligned}$$

It then follows that [88]

$$(p_i, Sp_j) = 0 \quad \text{for} \quad \mathrm{i} \neq \mathrm{j}$$

so that the search directions are (by definition) S orthogonal (conjugate) to each other. Also if y is the exact solution to Eqn. (C.19) then x_k minimizes [88]:

$$\| y - x \|$$

over the k dimensional affine space $x_0 + < p_0, p_1, \ldots p_{k-1} >$.

Again CG enjoys enormous advantages over GMRES in terms of storage and operation count when applied to symmetric positive definite linear systems. Unfortunately many of the advantages of CR and CG disappear when applied to nonsymmetric problems. Let G be a general (invertible) nonsymmetric matrix. Then

$$Gx = b \tag{C.22}$$

can be solved with CG or CR if one considers

$$G^* G x = G^* b \tag{C.23}$$

This involves the added expense and inconvenience of computing the adjoint operator G^*. The adjoints of any preconditioners must also be computed. This could

be nearly impossible if multigrid is used as a preconditioner. The elegant formulation shown in Eqn. (C.15) which allows GMRES to be immediately retro-fitted to existing codes is lost. Also the eigenvalues associated with Eqn. (C.23) are more spread out than those of Eqn. (C.22). Indeed the eigenvalues of G^*G are the squares of those of G. This means that more iterations are required to solve Eqn. (C.23) than to solve Eqn. (C.22). In fact conjugate gradient applied to Eqn. (C.23) with a basic underlying iterative method as preconditioner is often no faster than the basic underlying method, making acceleration of Eqn. (C.23) with any Krylov subspace method a hopeless cause. In view of the difficulties associated with applying CG or CR to Eqn. (C.23) one must ask how much does it really cost to apply GMRES to Eqn. (C.22) directly. In a typical application, the cost of GMRES turns out not to be too burdensome. The operations required by GMRES are fully vectorizable over vectors of length equal to the number of unknowns in the problem and are therefore capable of efficient implementation. For these reasons it is believed that in most cases applying GMRES to Eqn. (C.22) is preferable to using CG or CR on Eqn. (C.23).

Appendix D

POISSON SOLVER

D.1 SUMMARY OF THE POISSON SOLVER

The Poisson solver, denoted by T^{-1}, computes the perturbation potential ϕ from a given source function Q in a manner that automatically imposes the proper boundary condition at infinity. The computation of $T^{-1}(Q)$ is based on the discrete convolution, $G*Q$, of the sources Q with the discrete free space Green's function G. To get $T\phi = Q$, the function G must satisfy

$$(TG)(i,j,k) = \delta(i,j,k) = \begin{cases} 1 \text{ at } (0,0,0) \\ \\ 0 \text{ everywhere else} \end{cases} . \tag{D.1}$$

In addition G must satisfy a discrete far field boundary condition. It is sufficient to require that G be asymptotic to the continuous free space Green's function $-1/(4\pi r)$ as $r \to \infty$. In fact G may be approximated to arbitrary accuracy as $r \to \infty$ by an asymptotic expansion of the form

$$G(i,j,k) \sim -\frac{\Delta x \Delta y \Delta z}{4\pi}\left(\frac{1}{r} + \frac{\eta_3}{r^3} + \frac{\eta_5}{r^5} + \cdots\right). \tag{D.2}$$

Here $\eta_n = \eta_n(u,v,w,\Delta x,\Delta y,\Delta z)$ where

$$(u,v,w) = \left(\frac{i\Delta x}{r}, \frac{j\Delta y}{r}, \frac{k\Delta z}{r}\right) \text{ and } r = |(i\Delta x, j\Delta y, k\Delta z)|. \tag{D.3}$$

A general recursion formula (D.42) for the asymptotic coefficients η_n was derived after considerable effort. James' attempt [89] to derive the η_3 formula was consistent with the Poisson operator but not with the "recursion" relations (D.11) that G must satisfy. The MIT computer algebra program MACSYMA was helpful in evaluating η_5 and the corresponding downstream Green's function calculations.

The Poisson solver permits symmetry about the y and z planes $j = 0$ and $k = 0$ and automatically includes downstream sources. These are the sources on the downstream (m_x) plane of the computational box R. They are interpreted as extending to infinity in the x direction, so that wakes are automatically extended outside of R. A

corresponding downstream Green's function G_d, see (D.4) below, is computed along with the regular Green's function G. A correct solution requires that the downstream sources sum to zero; otherwise the solution is theoretically infinite everywhere, though in practice small deviations from the zero sum are tolerable.

The James algorithm, see section D.1.2 below, is used for the convolution. For an N^3 box, the real operation count of the TRANAIR implementation is $(116 + 10\log_2 N)N^3 + O(\log_2 N)N^2$. In practical cases the $O(\log_2 N)N^3$ asymptotic term is dominated by the large $O(N^3)$ term, and the $O(\log_2 N)N^2$ term is significant in small cases. As the result of a considerable programming effort, the code is typically 2 times faster than a comparable implementation of the standard convolution algorithm, whose operation count is $(72 + 35\log_2 N)N^3$. The Poisson solver achieved a rate of 480,000 grid points per second, or about 85 MFLOPS, on one processor of a Cray X-MP for a grid with dimensions $(m_x, m_y, m_z) = (160, 150, 27)$.

The memory requirement of the Poisson solver (including Green's function) is $N^3 + 51N^2$, which is N^3 asymptotically but closer to $2N^3$ in practice. This is due to the many scratch planes required for good vectorization of the dominant phase 2 of the algorithm. An earlier code, which was closer to James' technique, used much less memory in phase 2 but was much slower and more complicated. The standard algorithm uses $2N^3 + 23N^2$ words, which is asymptotically twice as much but is less than 1/3 more in typical cases.

Radix 2,3,5 FFT's (Fast Fourier Transforms) are used to achieve more flexibility in choosing (m_x, m_y, m_z) than the traditional radix 2 FFT, permitting a smaller computational box.

D.1.1 Summary of the Green's Function Algorithm

Buneman [90] found an analytical method for generating the 2D discrete Green's function in the case $\Delta x = \Delta y$. For the 3D case a new, semi-analytical method has been developed for arbitrary $\Delta x, \Delta y, \Delta z$. This method may be adapted to 2D and to higher dimensions as well.

The basic idea is to first compute Green's function data on the boundary of the box R and then solve for G in the interior by FFT techniques for a Poisson boundary value problem (see section D.3.1). Neumann boundary value data are used because a Neumann boundary value problem is solved by cosine transforms. For data that is symmetric about the origin on each axis, as it is for G, cosine transforms give the DFT of the solution. It is actually the DFT of G, denoted by $\hat{G}$, that is needed for the convolution. Thus by solving a Neumann boundary value problem, G itself need never be computed, only $\hat{G}$.

Symmetry determines the Neumann boundary data to be zero at the three boundary planes of R through the origin: $i = 0$, $j = 0$, and $k = 0$. The hard part is, of course, the boundary data for the three exterior planes $i = m_x$, $j = m_y$, $k = m_z$. The boundary data for the first of these planes, for example, must be computed as $(G(m_x + 1, j, k) - G(m_x - 1, j, k))/(2\Delta x)$, cosine transformed in j and k. If m_x is large, this may be computed accurately by the asymptotic expansion (D.2). For this reason the problem is first reformulated, if necessary, for a large box, whose dimen-

sions depend on the available scratch memory and on an attempt to equalize the radial dimensions $m_x\Delta x$, $m_y\Delta y$, and $m_z\Delta z$. Then this large box Neumann problem is partially solved to get the desired boundary data for the given box R.

The total CPU time for computing the Green's function data is less than .2 seconds using 100,000 words of scratch memory, corresponding to an operation count of 18 times the volume of the large box. This time is trivial since this data is computed only once and stored, to be read back at each subsequent call to the convolution algorithm. The data is accurate to about 9 or 10 significant digits, degrading only at high ratios of the deltas. The convolution algorithm preserves this accuracy.

The downstream Green's function G_d is defined by

$$G_d(i,j,k) = \sum_{\ell=m_x}^{\infty} G(\ell - i, j, k). \tag{D.4}$$

Formula (D.4) is derived from the requirement that if $Q(i,j,k) = 0$ for $i < m_x$ and $Q(i,j,k) = \tau(j,k)$ for $i \geq m_x$, then

$$(G * Q)(i,j,k) = (G_d(i,\cdot,\cdot) * \tau)(j,k).$$

The portion of each sum (D.4) outside the large box is computed by the Euler-Maclaurin formula for an infinite sum, with G evaluated by the expansion (D.2). Each such sum actually has an infinite part (see section D.2.4). But this part may be subtracted off because it is asymptotically equal for all (j,k) and will disappear upon convolution with the downstream sources, provided that they sum to zero.

The result of solving the Neumann problem for $\hat{G}$ is a simple algebraic formula in terms of three planes. These are the cosine transforms of the three exterior planes of boundary data (see section D.2.3). For $\hat{G}_d$, four planes are required since there is no symmetry in the x direction. Evaluation of these formulas typically adds about 5% to the CPU time of a convolution if there is no symmetry but saves two grid boxes of memory. If there is both y and z symmetry, the cost may rise to 25% of the CPU time but save eight grid boxes of memory.

D.1.2 Summary of the Convolution Algorithm

The procedure developed by R. A. James [89] is based on the decomposition of the perturbation potential ϕ into an "interior solution" θ and an "exterior solution" ψ, mediated by a "shielding charge" function σ. That is, write the Green's function solution $\phi = G * Q$ to the Poisson equation $T\phi = Q$ in the form

$$\phi = \theta + \psi$$

where θ and ψ are defined by

$$\begin{aligned} T\theta &= Q \quad \text{inside the box } R \text{ with} \\ \theta &= 0 \quad \text{on the boundary } \partial R \text{ and outside } R, \text{ and} \\ \psi &= G * \sigma \quad \text{for} \\ \sigma &= Q - T\theta \\ &\neq 0 \quad \text{only on } \partial R. \end{aligned}$$

Intuitively, the interior solution θ would result if the sources in R were shielded from everything outside by a wall at ∂R, for example. Subtracting σ from the given sources Q on ∂R gives induced sources, or charges, $T\theta$ on ∂R. By definition these charges have the same effect as a wall, hence the name shielding charge function. The exterior solution is so called because it incorporates the "exterior", or free-space, boundary condition.

A mathematical proof for the correctness of this decomposition is given by the following algebraic argument. Let θ and Q be functions defined on the 3D grid that are zero outside R and define $\sigma = Q - T\theta$ and $\psi = G * \sigma$. Then the discrete convolutions $G * Q$ and $G * \sigma$ are finite sums, and it is easy to verify that $G * (T\theta) = T(G * \theta)$. Hence

$$\phi = G * Q = G * (T\theta) + G * \sigma = T(G * \theta) + \psi = \theta + \psi. \qquad \text{(D.5)}$$

This algorithm involves three phases. For phase 1, solve $T\theta = Q$ inside R given $\theta = 0$ on ∂R and calculate $\sigma = Q - T\theta$, which is nonzero only on ∂R. For phase 2, compute $\psi = G * \sigma$ on ∂R by the standard convolution algorithm (described below). For phase 3, complete the computation of ψ by solving another Dirchlet Poisson problem: $T\psi = 0$ inside R given ψ on ∂R.

The efficiency of this algorithm comes in part from the fact that phase 2, $G * \sigma$, is much less work than $G * Q$ if done intelligently. Likewise, in phase 3 it is possible to take advantage of the zero interior sources. There is also a way to do the last part of the Dirichlet Poisson algorithm only once, instead of twice. Finally, the Dirichlet procedure is very efficient since it uses FFT sine transforms and a special tridiagonal solver (section D.3.1). Where there is a plane of symmetry, a mixed Dirichlet-Neumann problem results. This requires the use of "shifted sine or cosine" FFT transform algorithms to replace the sine transform (section D.3.2).

The standard FFT convolution procedure for $G * Q$ would be to apply FFT's in the x, y, and z directions to G and Q, multiply these complex results pointwise, then inverse transform. For this to be correct, both G and Q must be doubled in size along each axis, and this extension must be zero filled, with FFT's for these doubled lengths. The reason for this is that the FFT actually implements a "circular convolution"[91]. The zeros guarantee that this convolution equals the standard linear convolution.

The standard algorithm is adapted to phase 2 of the James algorithm in the following way. Since σ is restricted to ∂R, only the 6 boundary planes of R need be transformed directly. Then $\hat{\sigma}$ is constructed from these 6 planes, an xy plane (complex and quadrupled in size) at a time, along with the corresponding plane of $\hat{G}$. The values in these two planes are multiplied, and summations specified by the inverse transform formula are performed to get 6 output boundary planes. At the end, the inverses of the initial transformations are applied to these 6 planes. See section D.3.3 for explicit formulas for handling the 6 planes.

D.2 THEORY OF GREEN'S FUNCTION ALGORITHM

D.2.1 The Green's Function Definition

The Green's function G must satisfy the discrete Poisson equation

$$D^2G(i,j,k) = \delta(i,j,k) \tag{D.6}$$

where

$$\begin{aligned} D^2G(i,j,k) &= \frac{G(i+1,j,k) - 2G(i,j,k) + G(i-1,j,k)}{\Delta x^2} \\ &+ \frac{G(i,j+1,k) - 2G(i,j,k) + G(i,j-1,k)}{\Delta y^2} \\ &+ \frac{G(i,j,k+1) - 2G(i,j,k) + G(i,j,k-1)}{\Delta z^2}. \end{aligned}$$

In addition we assume that G satisfies a free space boundary condition of the form

$$G(i,j,k) = K\left(\frac{1}{r} + f(x,y,z)\right) \tag{D.7}$$

where K is a constant to be determined, $(x,y,z) = (i\Delta x, j\Delta y, k\Delta z)$, $r = |(x,y,z)|$, and f has continuous partial derivatives with asymptotic condition $r^2\nabla f \sim 0$ when $r \sim \infty$. Here "$x \sim 0$" means that x is infinitesimal (in the sense of nonstandard analysis[92]), "$x \sim \infty$" that $x^{-1} \sim 0$, and "$x \sim y$" that $x - y \sim 0$.

Mathematically it is not clear that a solution to (D.6) and (D.7) even exists, or if one exists, that it is unique. Therefore it is better to define G in a way that makes existence obvious, and then derive (D.6), (D.7), and show uniqueness. The form of the free space condition (D.7) permits a straightforward uniqueness proof (later in this section).

First define the discrete Green's function as the coefficients of a certain 3D Fourier series:

$$G(i,j,k) = \frac{1}{8\pi^3}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi} \frac{\exp\{(i\alpha + j\beta + k\gamma)i\}}{h(\alpha,\beta,\gamma)}\, d\alpha\, d\beta\, d\gamma \tag{D.8}$$

where

$$h(\alpha,\beta,\gamma) = -\frac{4}{\Delta x^2}\sin^2(\alpha/2) - \frac{4}{\Delta y^2}\sin^2(\beta/2) - \frac{4}{\Delta z^2}\sin^2(\gamma/2). \tag{D.9}$$

The function h^{-1} is singular at the origin, but this singularity is integrable by changing to spherical coordinates θ, ϕ, ρ, since the Jacobian is $O(\rho^2)$ and $h(\alpha,\beta,\gamma) = O(\rho^2)$ for $\rho \sim 0^+$. Therefore, according to Zygmund[93], the 3D Fourier series with coefficients G converges to h^{-1} in the L^1 norm, and also almost everywhere in, for example, the following form of "Abel convergence"

$$h^{-1}(\alpha,\beta,\gamma) = \lim_{r\to 1^-} \sum_{i,j,k} G(i,j,k)\, e^{(i\alpha+j\beta+k\gamma)i}\, r^{|i|+|j|+|k|}. \tag{D.10}$$

That the definition (D.8) satisfies the Poisson equation (D.6) is easily verified by calculations such as the following.

$$\begin{aligned}\exp(j+1)\beta i - 2\exp j\beta i + \exp(j-1)\beta i \\ &= (\exp\beta i + \exp(-\beta i) - 2)\exp j\beta i \\ &= 2(\cos(\beta)-1)\exp j\beta i \\ &= -4\sin^2(\beta/2)\exp j\beta i.\end{aligned}$$

Other important properties of (D.8) are the recursion relations:

$$\begin{aligned}\frac{G(i+1,j,k)-G(i-1,j,k)}{i\Delta x^2} &= \frac{G(i,j+1,k)-G(i,j-1,k)}{j\Delta y^2} \\ &= \frac{G(i,j,k+1)-G(i,j,k-1)}{k\Delta z^2}\end{aligned} \tag{D.11}$$

These are verified by using integration by parts to reduce each of the three differences to a common value. For example, the first difference in (D.11) may be integrated by parts in α to give

$$-\frac{1}{8\pi^3}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}\ln h(\alpha,\beta,\gamma)\,e^{(i\alpha+j\beta+k\gamma)i}\,d\alpha\,d\beta\,d\gamma$$

since $\exp(i+1)\alpha i - \exp(i-1)\alpha i = 2i\sin(\alpha)\exp i\alpha i = -\Delta x^2 i h_\alpha \exp i\alpha i$ from (D.9). These relations are first order finite difference analogs of the differential relations

$$\frac{1}{x}\left(\frac{1}{r}\right)_x = \frac{1}{y}\left(\frac{1}{r}\right)_y = \frac{1}{z}\left(\frac{1}{r}\right)_z.$$

If the definition (D.8) is interpreted as an iterated integral, one of the three integrals may be done analytically. For example, suppose that $k > 0$ and do the γ integral by writing it as a contour integral along the infinite rectangle in the upper half complex plane from $-\pi + \infty i$ to $-\pi$ to $+\pi$ to $+\pi + \infty i$ and back. Except for $\alpha = \beta = 0$, there will be exactly one pole inside the contour. This is at the point $\gamma_0 i$ which satisfies $h(\alpha,\beta,\gamma_0 i) = 0$. Its residue is given by

$$\frac{1}{h_\gamma(\alpha,\beta,\gamma_0 i)} = \frac{\Delta z^2 i}{2\sinh(\gamma_0)}$$

where (D.9) shows that $\gamma_0 = \gamma_0(\alpha,\beta)$ is defined by

$$\sinh^2(\gamma_0(\alpha,\beta)/2) = \frac{\Delta z^2}{\Delta x^2}\sin^2(\alpha/2) + \frac{\Delta z^2}{\Delta y^2}\sin^2(\beta/2). \tag{D.12}$$

Therefore the triple integral in (D.8) may be reduced to the double integral

$$\begin{aligned}G(i,j,k) &= -\frac{\Delta z^2}{8\pi^2}\int_{-\pi}^{\pi}\int_{-\pi}^{\pi}\frac{\exp\{(i\alpha+j\beta)i - k\gamma_0(\alpha,\beta)\}}{\sinh(\gamma_0(\alpha,\beta))}\,d\alpha\,d\beta \\ &= -\frac{\Delta x\Delta y\Delta z}{8\pi^2}\int_{-\pi/\Delta x}^{\pi/\Delta x}\int_{-\pi/\Delta y}^{\pi/\Delta y}\frac{\exp\{((u\chi+v\psi)i - w\zeta)r\}}{\sinh(\gamma)/\Delta z}\,d\chi\,d\psi\end{aligned} \tag{D.13}$$

where $(u, v, w) = (x/r,\ y/r,\ z/r)$, $\gamma = \gamma_0(\alpha, \beta)$, and $(\chi, \psi, \zeta) = (\alpha/\Delta x, \beta/\Delta y, \gamma/\Delta z)$ are normalized coordinates. Numerical integration of (D.13) is possible, but expensive for large i, j, or k.

That the free space condition (D.7) is satisfied by definition (D.8) follows from the development of the asymptotic expansion of (D.13) later in section D.2.2. It is already obvious from (D.8) that G is infinitely differentiable as a function of x, y, z so that the function f in (D.7) has the required smoothness.

Now let us present the uniqueness proof for condition (D.7). The proof is based on the discrete divergence theorem. Let $\mathbf{g} = (g_1, g_2, g_3)$ be a function defined on a grid box $R = [b_x, e_x] \times [b_y, e_y] \times [b_z, e_z]$. Also let $D_f = (D_{f1}, D_{f2}, D_{f3})$ denote the forward difference operators in x, y, z; e.g., $(D_{f1})g_1(i, j, k) = (g_1(i+1, j, k) - g_1(i, j, k))/\Delta x$. Then just do all possible cancellations to get

$$\begin{aligned}\sum_{i,j,k \in R^0} (D_f \cdot \mathbf{g})(i,j,k)\,\Delta x \Delta y \Delta z = \\ \sum_{j,k \in R^0_{yz}} (g_1(e_x, j, k) - g_1(b_x + 1, j, k))\,\Delta y \Delta z \\ + \sum_{i,k \in R^0_{xz}} (g_2(i, e_y, k) - g_2(i, b_y + 1, k))\,\Delta x \Delta z \\ + \sum_{i,j \in R^0_{xy}} (g_3(i, j, e_z) - g_3(i, j, b_z + 1))\,\Delta x \Delta y\end{aligned}$$

where $R^0 = [b_x + 1, e_x - 1] \times [b_y + 1, e_y - 1] \times [b_z + 1, e_z - 1]$, etc. This statement may be abbreviated by letting ΔS represent the boundary deltas, $\Delta R = \Delta x \Delta y \Delta z$, and $\mathbf{n}$ = the outward unit boundary normal vector, to get

$$\sum_{R^0} D_f \cdot g\,\Delta R = \sum_{\partial R^0} g \cdot \mathbf{n}\,\Delta S. \tag{D.14}$$

Now let $(e_x, e_y, e_z) = -(b_x, b_y, b_z) = (n, n, n)$ and apply (D.14) to the backward difference operators $D_b G = (D_{b1}, D_{b2}, D_{b3})G$, defined by $(D_{b1}G)(i, j, k) = (G(i, j, k) - G(i-1, j, k))/\Delta x$, etc. By (D.6) and (D.14) the result is

$$\begin{aligned}\Delta R = \sum_{R^0} D^2 G\,\Delta R = \sum_{\partial R^0} D_b G \cdot \mathbf{n}\,\Delta S \\ = 8 \Bigg[\sum_{j=0}^{n-1}{}' \sum_{k=0}^{n-1}{}' (D_{b1}G)(n, j, k)\,\Delta y \Delta z \\ + \sum_{i=0}^{n-1}{}' \sum_{k=0}^{n-1}{}' (D_{b2}G)(i, n, k)\,\Delta x \Delta z \\ + \sum_{i=0}^{n-1}{}' \sum_{j=0}^{n-1}{}' (D_{b3}G)(i, j, n)\,\Delta x \Delta y \Bigg].\end{aligned}$$

where $\sum'$ denote a summation with a factor of $\frac{1}{2}$ on the summand for the xxxx xxxx of

integration. Next apply Taylor's theorem with remainder to (D.7) to get, for example,

$$(D_{b1}G)(n,j,k) = K\left(-\frac{u}{r^2} + \nabla f\right)$$

where the right side is evaluated at some point between $(n-1,j,k)$ and (n,j,k). When summed over the boundary the ∇f term drops out for $n \sim \infty$, since $r = O(n)$ and $r^2\nabla f \sim 0$ by assumption. That is, the constant K is determined independently of f by summing the $O(1/r^2)$ terms over the boundary.

Next we show that f in (D.7) is uniquely determined by another application of the discrete divergence theorem. Suppose that f and f' are two different functions satisfying (D.7) and let $g = f - f'$. If $D_f g = 0$ everywhere, then g must be constant, and by the $r^2\nabla f \sim 0$ condition this constant must be zero. Therefore we may assume, for example, that $(D_{f1}g)(i^*,j^*,k^*) \not\sim 0$ and define $h(i,j,k) = 0$ if $i \le i^*$ and $h(i,j,k) = \Delta x$ if $i \ge i^* + 1$. Now apply (D.14) to $(D_b g)h$ to get

$$\sum_{R^0} D_f \cdot [(D_b g)h]\Delta R = \sum_{\partial R^0} h(D_b g)\cdot \mathbf{n}\Delta s \sim 0$$

since $r^2 D_b g \sim 0$. On the other hand, simple algebra gives the following product rule for discrete differentiation

$$\begin{aligned}\sum_{R^0} D_f \cdot [(D_b g)h]\,\Delta R &= \sum_{R^0}(D^2 g)h\,\Delta R + \sum_{R^0}(D_f g)\cdot(D_f h)\,\Delta R \\ &= 0 + (D_{f1}g)(i^*,j^*,k^*)\,\Delta R \not\sim 0.\end{aligned}$$

This contradiction proves that $g \equiv 0$, hence the uniqueness of f.

There is a condition equivalent to the traditional free space condition (D.7) that is more natural from a mathematical point of view. Simply assume that the Fourier series (D.10) converges in the L^1 norm. Then we demonstrate that the full series converges in L^1 to h^{-1}, hence that (D.8) holds by Fourier series inversion. For notational convenience, the argument is illustrated in the y dimension, though at least three dimensions are required for correctness. First scale the Poisson equation by $r^{|j|}e^{j\beta i}$, and sum over j, using (D.6),(D.10), G symmetry, and trig identities to get

$$\begin{aligned}1 &= \sum_{j=-\infty}^{\infty}(D^2G)(j)\,r^{|j|}\,e^{j\beta i} \\ &= 2(G(1)-G(0)) + 2\sum_{j=1}^{\infty}(G(j-1)-2G(j)+G(j+1))r^j\cos j\beta \\ &= 2\left[G(1) - g(r,\beta) + \sum_{j=0}^{\infty}G(j)r^{j+1}\cos(j+1)\beta + \sum_{j=2}^{\infty}G(j)r^{j-1}\cos(j-1)\beta\right] \\ &= \left((r+r^{-1})\cos\beta - 2\right)g(r,\beta) + (r^{-1}-r)(\sin\beta)\tilde{g}(r,\beta) + (r-r^{-1})(\cos\beta)G(0)\end{aligned}$$

where $g(r,\beta) = \sum_j G(j)r^{|j|}e^{j\beta i}$ and $\tilde{g}(r,\beta) = 2\sum_{j=1}^{\infty}G(j)r^j\sin j\beta$. By assumption both $g(r,y)$ and $\tilde{g}(r,y)$ converge in L^1 to limits $g(\beta)$ and $\tilde{g}(\beta)$ as $r \to 1^-$, so g satisfies $1 = 2(\cos(\beta)-1)g(\beta) = -4\sin^2(\beta/2)g(\beta)$ almost everywhere.

It can also be demonstrated directly from corresponding Fourier integral results that all terms of the asymptotic expansion (D.2) have Fourier series, in the sense of (D.10), that are L^1 summable.

D.2.2 The Asymptotic Expansion

There are two ways of deriving the asymptotic expansion. One way is to apply the appropriate integration techniques directly to the double integral (D.13) above. Another way is to assume the form of the expansion, plug this into the Poisson equation (D.6) and the recursion relations (D.11), and use Taylor series.

The first approach demonstrates the existence of an asymptotic expansion of the Fourier series definition (D.8). In particular, it establishes the free space condition (D.7) and the value of the scale factor. Then by the uniqueness argument in section D.2.1 above, the second approach is justified. The latter method is algebraically easier.

First Approach

In the first approach, the χ, ψ coordinates of the double integral (D.13) are transformed to polar coordinates θ, ρ. Asymptotic evaluation of (D.13) reduces to the computation of

$$I(x,y,z) = \int_0^{2\pi} \int_0^{\rho^*} \frac{e^{-(w\zeta(\rho,\theta)-q(\theta)\rho i)r}}{\sinh(\gamma(\rho,\theta))/\Delta z}\, \rho\, d\rho\, d\theta \tag{D.15}$$

where $r \sim \infty$, $\rho^* > 0$ with $\rho^* \not\sim 0$, and $q(\theta) = u\cos(\theta) + v\sin(\theta)$. That is, $r^n\,(G(i,j,k) + (\Delta x \Delta y \Delta z/8\pi^2) I(x,y,z)) \sim 0$ for all finite $n > 0$. Note that $w > 0$ was assumed in (D.13).

To evaluate the inner integral, change the radial coordinate ρ to the complex coordinate

$$\begin{aligned} \nu &= \nu(\rho,\theta) = w\zeta - q\rho i \\ d\nu &= (w\zeta_\rho - qi)d\rho. \end{aligned} \tag{D.16}$$

To get ζ_ρ, differentiate (D.12) and use the notation $s_z = \sinh(\gamma)/\Delta z$, $c_z = \cosh(\gamma)$, $s_x = \sin(\alpha)/\Delta x$, $c_x = \cos(\alpha)$, $s_y = \sin(\beta)/\Delta y$, $c_y = \cos(\beta)$, $c_\theta = \cos(\theta)$, $s_\theta = \sin(\theta)$, $t(\rho,\theta) = s_x c_\theta + s_y s_\theta$, and $D(\rho,\theta) = wt - s_z qi$. Then

$$\zeta_\rho = \frac{t}{s_z}, \quad \rho'(\nu) = \frac{s_z}{D}, \quad \zeta'(\nu) = \frac{t}{D}. \tag{D.17}$$

Thus from (D.15)

$$I = \int_0^{2\pi} \int_0^{\nu^*} e^{-\nu r} f_\theta(\nu)\, d\nu\, d\theta \quad \text{for} \quad f_\theta(\nu) = \frac{\rho}{D}. \tag{D.18}$$

Now the asymptotic expansion is obtained by applying integration by parts repeatedly to the inner integral of (D.18):

$$I = \sum_{n=1}^{N} \left(\int_0^{2\pi} f_\theta^{(n-1)}(0^+)\, d\theta \right) \frac{1}{r^n} + R_m \tag{D.19}$$

with remainder

$$R_m = \left(\int_0^{2\pi} \int_0^{\nu^*} e^{-\nu r} f_\theta^{(N)}(\nu)\, d\nu\, d\theta \right) \frac{1}{r^N}$$

and $f^{(n)}(0^+) \equiv \lim_{\rho \to 0+} f^{(n)}(\nu(\rho))$. It is not immediately clear that any terms of the expansion (D.19) are finite since it has not yet been shown that the limits $f_\theta^{(n)}(0^+)$ exist and are integrable in θ. In fact we will demonstrate that $f_\theta^{(n)}(\nu)$ is uniformly bounded on $[0, 2\pi] \times [0, \nu^*]$ by showing how to compute the limits. Thus $R_m = O(1/r^{N+1})$.

Let us begin by calculating the first asymptotic term to get the constant K in (D.7). Note that for $\rho \sim 0^+$

$$(f(\nu))^{-1} = w\frac{t}{\rho} - \frac{s_z}{\rho} qi \sim w(c_\theta^2 + s_\theta^2) - \frac{\zeta}{\rho} qi \sim w - qi \tag{D.20}$$

from (D.18), (D.17), and (D.12). Next note that $q(\theta) = (u^2 + v^2)^{1/2} \cos(\theta - \theta_0) = (1 - w^2)^{1/2} \cos(\theta - \theta_0)$ for $\theta_0 = \tan^{-1}(v/u)$. Therefore

$$\begin{aligned}
\int_0^{2\pi} f_\theta(0^+) d\theta &= \int_0^{2\pi} \frac{w + q(\theta) i}{w^2 + q^2(\theta)}\, d\theta \\
&= \int_0^{2\pi} \frac{w + \cos(\theta)(1 - w^2)^{1/2} i}{w^2 + (1 - w^2)\cos^2(\theta)}\, d\theta \\
&= 4w \int_0^{\pi/2} \frac{d\theta}{\cos^2(\theta) + w^2 \sin^2(\theta)} \\
&= 4w \int_0^{\pi/2} \frac{\sec^2(\theta)\, d\theta}{1 + w^2 \tan^2(\theta)} \\
&= 4 \int_0^{\infty} \frac{dx}{1 + x^2} = 2\pi
\end{aligned} \tag{D.21}$$

so from (D.13)

$$K = -\frac{\Delta x \Delta y \Delta z}{4\pi}.$$

Next consider the first derivative term. Use the notation $W = w - qi$, so that $D/\rho \sim W$ for $\rho \sim 0^+$ by (D.20). Then

$$f' = \frac{\rho'}{D} - \frac{\rho D'}{D^2} = \frac{\rho' - f D'}{D} \tag{D.22}$$

and by (D.17)

$$\begin{aligned}
\rho' &\sim W^{-1} \\
t' &= \left(c_x c_\theta^2 + c_y s_\theta^2 \right) \rho' \sim \rho' \sim W^{-1} \\
\zeta' &\sim (c_\theta^2 + s_\theta^2)/(D/\rho) \sim W^{-1}
\end{aligned}$$

so

$$\begin{aligned}
D' = w t' - c_z \zeta' qi &\sim (w - qi)/W = 1 \\
\rho' - f D' &\sim W^{-1} - W^{-1} \sim 0.
\end{aligned} \tag{D.23}$$

Therefore a version of L'Hospital's rule may be applied to evaluate (D.22):

$$f' \sim \frac{\rho'' - fD'' - f'D'}{D'} \quad \text{or} \quad f' \sim \frac{1}{2}(\rho'' - fD''). \tag{D.24}$$

This version may be stated in the following way. If $h(x)g(x) = f(x)$ for g and f continuous on $[0, x^*]$ with $g(0) = f(0) = 0$, and f and g continuously differentiable on $(0, x^*)$ with $f'(x) = f_1(x) + f_2(x)h(x)$ and such that the limits $g'(0^+)$, $f_1(0^+)$, $f_2(0^+)$ exist and $g'(0^+) - f_2(0^+) \neq 0$, then $h(0^+)$ exists and equals $f_1(0^+)/(g'(0^+) - f_2(0^+))$. The usual proof by the mean value theorem applies.

Now by (D.23)

$$\begin{aligned} D'' &= wt'' - s_z'' qi \\ t'' &= -\left(s_x \Delta x^2 c_\theta^3 + s_y \Delta y^2 s_\theta^3\right)(\rho')^2 + \left(c_x c_\theta^2 + c_y s_\theta^2\right)\rho'' \sim \rho'' \\ s_z'' &= -s_z \Delta z^2 (\zeta')^2 + c_z \zeta'' \sim \zeta''. \end{aligned} \tag{D.25}$$

Using (D.17), (D.23), and (D.25) and applying L'Hospital's rule again leads to simultaneous equations for ρ'' and ζ'':

$$\begin{aligned} \rho'' &\sim (s_z'' - \rho' D'' - \rho'' D')/D' \\ &\sim \zeta'' - W^{-1}(w\rho'' - \zeta'' qi) - \rho'' \\ \zeta'' &\sim (t'' - \zeta' D'' - \zeta'' D')/D' \\ &\sim \rho'' - W^{-1}(w\rho'' - \zeta'' qi) - \zeta'' \end{aligned} \tag{D.26}$$

Since these equations are homogeneous, they may be solved to get $\rho'' \sim \zeta'' \sim 0$. Hence $f_\theta'(0^+) \equiv 0$ by (D.24) and (D.25). To solve for $f_\theta^{(n)}(0^+)$ when $n \geq 2$, simply generalize the method used for the first derivative term. Assume by induction that the limits exist at $\nu = 0$ for $f^{(j)}$, $\rho^{(j+1)}$, and $\zeta^{(j+1)}$ when $j = 0, ..., n-1$. Then the $t^{(j+1)}$ and $D^{(j+1)}$ limits also exist, as in (D.25). To derive the formula for $\rho^{(n)}$, note that

$$\rho^{(n)} = (fD)^{(n)} = \sum_{j=0}^{n} \binom{n}{j} f^{(j)} D^{(n-j)}$$

so the $f^{(n)}D$ limit exists and

$$f^{(n)}D = \rho^{(n)} - F_n - nf^{(n-1)}D' \tag{D.27}$$

for

$$F_n = \sum_{j=0}^{n-2} \binom{n}{j} f^{(j)} D^{(n-j)}.$$

Also assume by induction that the $f^{(n)}D$ limit is zero, already verified by (D.22) and (D.23) for $n = 1$. Then by L'Hospital's rule

$$f^{(n)}D' \sim \rho^{(n+1)} - F_n' - nf^{(n-1)}D'' - nf^{(n)}D'$$

so

$$\rho^{(n+1)} - F_{n+1} - (n+1)f^{(n)}D' \sim 0,$$

verifying the $f^{(n)}D$ induction hypothesis and the formula

$$f^{(n)} \sim \frac{1}{n+1}\left(\rho^{(n+1)} - F_{n+1}\right) \tag{D.28}$$

provided that the existence of the $\rho^{(n+1)}$ and $\zeta^{(n+1)}$ limits has been demonstrated.

To calculate $\rho^{(n+1)}$ and $\zeta^{(n+1)}$, just mimic the argument used in (D.27) - (D.28), except that you end up with a pair of simultaneous equations as in (D.26). First expand $s_z^{(n)} = (\rho' D)^{(n)}$ and $t^{(n)} = (\zeta' D)^{(n)}$ and assume by induction that the $\rho^{(n+1)}D$ and $\zeta^{(n+1)}D$ limits are zero. Then generalize the version of L'Hospital's rule cited above to the case of two simultaneous limits to get

$$\begin{aligned} s_z^{(n+1)} - \rho' D^{(n+1)} - P_{n+1} - (n+1)\rho^{(n+1)} &\sim 0 \\ t^{(n+1)} - \zeta' D^{(n+1)} - Q_{n+1} - (n+1)\zeta^{(n+1)} &\sim 0 \end{aligned} \tag{D.29}$$

where

$$P_{n+1} = \sum_{j=1}^{n-1} \binom{n+1}{j} \rho^{(j+1)} D^{(n+1-j)}\ ,\ Q_{n+1} = \sum_{j=1}^{n-1} \binom{n+1}{j} \zeta^{(j+1)} D^{(n+1-j)}.$$

Evaluating $D^{(n+1)} \sim wt^{(n+1)} - s_z^{(n+1)}qi$, $s_z^{(n+1)} \sim S_{n+1} + \zeta^{(n+1)}$, and $t^{(n+1)} \sim T_{n+1} + \rho^{(n+1)}$ separates out all the remaining $\rho^{(n+1)}$ and $\zeta^{(n+1)}$ terms. Now solve (D.29) by collecting these terms:

$$\begin{aligned} -\left(n+1+\frac{w}{W}\right)\rho^{(n+1)} + \left(1+\frac{qi}{W}\right)\zeta^{(n+1)} &\sim P_{n+1} + \frac{w}{W}T_{n+1} - \left(1+\frac{qi}{W}\right)S_{n+1} \\ \left(1-\frac{w}{W}\right)\rho^{(n+1)} - \left(n+1-\frac{qi}{W}\right)\zeta^{(n+1)} &\sim Q_{n+1} - \left(1-\frac{w}{W}\right)T_{n+1} - \frac{qi}{W}S_{n+1} \end{aligned}$$

or

$$\begin{aligned} \rho^{(n+1)} &\sim \left[(n+1)w\left(S_{n+1} - T_{n+1}\right) - wQ_{n+1} - \left((n+1)W - qi\right)P_{n+1}\right]/DD \\ \zeta^{(n+1)} &\sim \left[(n+1)qi\left(S_{n+1} - T_{n+1}\right) + qiP_{n+1} - \left((n+1)W + w\right)Q_{n+1}\right]/DD \end{aligned} \tag{D.30}$$

for $DD = (n+1)(n+2)W$.

But $W(\theta)$ is uniformly bounded away from zero by the assumption that $w > 0$. Therefore the use of the simultaneous limit version of L'Hospital's rule is valid for all θ and all the induction hypotheses are verified. Also the denominator of $f_\theta^{(n)}(0^+)$ is a power of W, so $f_\theta^{(n)}(\nu)$ must be uniformly bounded on $[0, 2\pi] \times [0, \nu^*]$. This fully establishes the both the finiteness and computability of the terms of the expansion (D.19) and the $O(1/r^{N+1})$ property of its remainder. A simple consequence is the free space condition (D.7). However the method used in the next section provides a simpler way to derive convenient formulas for the higher order terms.

Second Approach

In the second approach it is assumed that the expansion has the form

$$G(i,j,k) \sim -\frac{\Delta x \Delta y \Delta z}{4\pi}\left(\frac{\eta_1}{r}+\frac{\eta_2}{r^2}+\frac{\eta_3}{r^3}+\cdots\right) \tag{D.31}$$

where $\eta_n = \eta_n(\mathbf{u}, \Delta x, \Delta y, \Delta z)$, $\mathbf{u} = (u,v,w) = (x/r, y/r, z/r)$, and $(x,y,z) = (i\Delta x, j\Delta y, k\Delta z)$. G is assumed to be a smooth function of x,y,z and each η_n a smooth function of u,v,w.

The expansion (D.31) is substituted into the Poisson equation (D.6) and recursion relations (D.11). But first all terms, except the $G(i,j,k)$ terms, are expanded in Taylor series about (x,y,z). For example,

$$G(i+1,j,k) = G(i,j,k) + \sum_{n=1}^{\infty} D_x^n G(i,j,k)\frac{\Delta x^n}{n!} \tag{D.32}$$

where $(D_x, D_y, D_z) = \nabla$ is the gradient operator. For each equation all terms that have the same power of $1/r$ are collected and set to zero. For this purpose, it is easily demonstrated by induction that $D_x^n(\eta_\ell/r^\ell) = O(1/r^{n+\ell})$.

Certain auxiliary equations are also necessary. This includes the spherical equation

$$u^2 + v^2 + w^2 = 1 \tag{D.33}$$

and orthogonality relations of the form $F_r = \nabla F \cdot \mathbf{u} = 0$ for $F(x,y,z) = F(ur, vr, wr)$ a function whose value is independent of the scale factor r for $r \neq 0$. In mathematical terminology, F is a function on the real projective plane. For example,

$$(\eta_n)_r = \nabla \eta_n \cdot \mathbf{u} = 0. \tag{D.34}$$

The $1/r^{m+1}$ equations, $m \geq 1$, that come from the recursion relations are

$$\begin{aligned} vD_x\left(\frac{\eta_m}{r^m}\right) + v\frac{q_{m1}}{r^m} &= uD_y\left(\frac{\eta_m}{r^m}\right) + u\frac{q_{m2}}{r^m} \\ wD_y\left(\frac{\eta_m}{r^m}\right) + w\frac{q_{m2}}{r^m} &= vD_z\left(\frac{\eta_m}{r^m}\right) + v\frac{q_{m3}}{r^m} \end{aligned} \tag{D.35}$$

where $\mathbf{q_m} = (q_{m1}, q_{m2}, q_{m3})$ is defined by

$$\mathbf{q}_m = r^m \sum_{\ell=1}^{k_m} \frac{\mathbf{s}_{m\ell}}{(2\ell+1)!} \quad \text{for } k_m = \text{ integerpartof}\left[\frac{m-1}{2}\right], \tag{D.36}$$

$$\mathbf{s}_{m\ell} = D^{2\ell}(2\ell+1)\left(\frac{\eta_{m-2\ell}}{r^{m-2\ell}}\right), \quad D^i(j) = \left(\Delta x^i D_x^j, \Delta y^i D_y^j, \Delta z^i D_z^j\right). \tag{D.37}$$

Equations (D.34) - (D.35) evaluate to a 3 by 3 system of equations in $\nabla\eta_m$, which is solved to get

$$\nabla \eta_m = \mathbf{u}\mathbf{u}^t \mathbf{q}_m - \mathbf{q}_m. \tag{D.38}$$

Thus $\nabla\eta_1 = \mathbf{0}$, so η_1 is constant. According to section D.2.2 this constant value is 1. The $1/r^{m+2}$ equation that comes from the Poisson equation is

$$\frac{1}{2}\nabla^2\left(\frac{\eta_m}{r^m}\right) + \sum_{\ell=1}^{k_m}\frac{\nabla\cdot\mathbf{s}_{m\ell}}{(2\ell+2)!} = 0. \tag{D.39}$$

Using (D.33) and (D.34) to simplify (D.39) yields

$$\eta_m = -\frac{r^2}{(m-1)m}\left(\nabla^2\eta_m + 2r^m\sum_{\ell=1}^{k_m}\frac{\nabla\cdot\mathbf{s}_{m\ell}}{(2\ell+2)!}\right). \tag{D.40}$$

Now (D.38), (D.36), and (D.39) yield

$$\begin{aligned}\nabla^2\eta_m &= \frac{m}{r}(\nabla\eta_m\cdot\mathbf{u}) + r^m\sum_{\ell=1}^{k_m}\frac{\nabla\cdot(\mathbf{u}\mathbf{u}^t\mathbf{s}_{m\ell}) - \nabla\cdot\mathbf{s}_{m\ell}}{(2\ell+1)!}\\ &= r^m\sum_{\ell=1}^{k_m}\frac{-\nabla\cdot\mathbf{s}_{m\ell} + \mathbf{u}^t\nabla\mathbf{s}_{m\ell}\mathbf{u} + 2\mathbf{u}\cdot\mathbf{s}_{m\ell}/r}{(2\ell+1)!}\\ &= -r^{m-1}\sum_{\ell=1}^{k_m}\frac{r\nabla\cdot\mathbf{s}_{m\ell} + (m-1)\mathbf{u}\cdot\mathbf{s}_{m\ell}}{(2\ell+1)!},\end{aligned} \tag{D.41}$$

using the orthogonality relation $\nabla(r^{m+1}\mathbf{s}_{m\ell})\cdot\mathbf{u} = \mathbf{0}$ to get $\nabla\mathbf{s}_{m\ell}\mathbf{u} = -(m+1)\mathbf{s}_{m\ell}/r$.

Thus substituting (D.41) into (D.40) gives the following recursion formula for the asymptotic coefficients when $m \geq 2$.

$$\eta_m = \sum_{\ell=1}^{k_m}\frac{r^{m+1}}{(2\ell+1)!m}\left(\mathbf{u} + \frac{\ell r}{(\ell+1)(m-1)}\nabla\right)\cdot\mathbf{s}_{m\ell} \tag{D.42}$$

It is immediate from (D.42) that $\eta_2 = 0$. Also $\mathbf{s}_{m\ell}$ involves only lower even asymptotic coefficients η_n if m is even, so (D.42) gives a proof by induction that $\eta_m = 0$ for all even m. In addition, it is shown below that η_{2k+1} may always be expressed as a polynomial in factors of the form $U^{2i}(2j)$, as in (D.43) and (D.44):

$$\eta_3 = \frac{1}{8}U^2(0) - \frac{3}{4}U^2(2) + \frac{5}{8}U^2(4) \tag{D.43}$$

$$\begin{aligned}\eta_5 &= \frac{3}{16}U^4(0) - \frac{65}{16}U^4(2) + \frac{105}{8}U^4(4) - \frac{189}{16}U^4(6)\\ &+ \left(\frac{3}{128}U^2(0) - \frac{15}{32}U^2(2) + \frac{35}{64}U^2(4)\right)U^2(0)\\ &+ \left(\frac{105}{32}U^2(2) - \frac{315}{32}U^2(4)\right)U^2(2) + \frac{1155}{128}U^2(4)U^2(4)\end{aligned} \tag{D.44}$$

where $U^i(j) = \Delta x^i u^j + \Delta y^i v^j + \Delta z^i w^j$. These formulas for η_3 and η_5 are good for computation as well as compactness of expression.

First, it is clear by induction from (D.42) that η_{2k+1} is always a homogeneous polynomial in Δx, Δy, Δz of degree $2k$, that the coefficients of this polynomial are

themselves polynomials in u, v, w, and that all powers of $u, v, w, \Delta x, \Delta y, \Delta z$ are even. The way to get the special polynomial form used in (D.43) and (D.44) is as follows.

The basic idea is to first compute the x partial derivatives of u^j/r^i in the form

$$D_x^k\left(\frac{u^j}{r^i}\right) = D_x^k\left(\frac{x^j}{r^{i+j}}\right) = \sum_{n=0}^{k} a_{j,i}^k(n)\frac{x^{j-k+2n}}{r^{i+j+2n}} \tag{D.45}$$

with $a_{j,i}^k(n) = 0$ if $j-k+2n < 0$. The same coefficients are valid for the corresponding expansions of the y partials of v^j/r^i and the z partials of w^j/r^i. Next extend (D.45) to

$$\begin{aligned} D_x^k\left(\frac{U^m(j)}{r^i}\right) &= \sum_{n=0}^{k}\left[a_{j,i}^k(n)\frac{\Delta x^m x^{j-k+2n}}{r^{i+j+2n}} + a_{0,i+j}^k(n)\frac{\left(\Delta y^m y^j + \Delta z^m z^j\right)x^{-k+2n}}{r^{i+j+2n}}\right] \\ &= \sum_{n=0}^{k}\left[\left(a_{j,i}^k(n) - a_{0,i+j}^k\right)\frac{\Delta x^m u_{j-k+2n}}{r^{i+k}} + U^m(j)\frac{u_{-k+2n}}{r^{i+k}}\right] \end{aligned}$$

This formula may be extended further to compute the coefficient η_{2k+1} by the following method.

$$r^{2k+3}\nabla\cdot D^{2\ell}(2\ell+1)\left[\frac{U^{2k-2\ell}(j)}{r^{2k+1-2\ell}}\right] =$$

$$\sum_{n=0}^{m_\ell}\left[\left(a_{j,m_k}^{m_\ell}(n) - a_{0,m_k}^{m_\ell}(n)\right)U^{2k}(j+m_n) + a_{0,m_k}^{m_\ell}(n)U^{2\ell}(m_n)U^{2k-2\ell}(j)\right] \tag{D.46}$$

where $m_\ell = 2\ell+2$, $m_n = 2n-2\ell-2$, and $m_k = 2k+1-2\ell+j$. Exactly the same formula is used to compute $r^{2k+2}\mathbf{u}\cdot D^{2\ell}(2\ell+1)\left[U^{2k-2\ell}(j)/r^{2k+1-2\ell}\right]$, except that m_ℓ is changed to $2\ell+1$ and m_n to $2n-2\ell$. Thus the η_5 formula (D.44) is obtained from the η_3 formula (D.43) by using (D.46) to evaluate the recursion formula (D.42).

To get η_7 one must in addition evaluate the same operators on terms of the form $U^{i_1}(j_1)U^{i_2}(j_2)/r^{2k+1-2\ell}$ where $i_1 + i_2 = 2k-2\ell$. The result is

$$\begin{aligned} \sum_{n=0}^{m_\ell}&\left(a_{j_1+j_2,m_k}^{m_\ell}(n) - a_{j_1,m_k}^{m_\ell}(n) - a_{j_2,m_k}^{m_\ell}(n) + a_{0,m_k}^{m_\ell}(n)\right)U^{2k}(j_1+j_2+m_n) \\ &+ \left(a_{j_1,m_k}^{m_\ell}(n) - a_{0,m_k}^{m_\ell}(n)\right)U^{i_1+2\ell}(j_1+m_n)U^{i_2}(j_2) \\ &+ \left(a_{j_2,m_k}^{m_\ell}(n) - a_{0,m_k}^{m_\ell}(n)\right)U^{i_2+2\ell}(j_2+m_n)U^{i_1}(j_1) \\ &+ a_{0,m_k}^{m_\ell}(n)U^{2\ell}(m_n)U^{i_1}(j_1)U^{i_2}(j_2) \end{aligned} \tag{D.47}$$

with m_k, m_ℓ, and m_n exactly as before.

Formula (2.41) generalizes to $U^i(j)$ products of arbitrary length by using the input-output formula of combinatorial set theory.

D.2.3 The Three Plane Representation

As described in section D.1.1, the Green's function G is computed from its asymptotic values by solving a Neumann boundary value problem with a delta source function.

The theory of FFT solutions to Neumann problems is given in section D.3.1. Here that theory is applied to get a simple formula, (D.51) below, for the DFT, or discrete Fourier transform, $\hat{G}$, defined by (D.90), in terms of the boundary data.

The theory begins by transforming the boundary data into equivalent boundary sources, as in section D.3.1. Thus the sources ρ are all zero except for $\rho(0,0,0) = 1$, $\rho(m_x, j, k) = g_x(j,k)$, $\rho(i, m_y, k) = g_y(i,k)$, and $\rho(i, j, m_z) = g_z(i,j)$, where

$$g_x(j,k) = -\frac{G(m_x+1,j,k) - G(m_x-1,j,k)}{\Delta x^2} \tag{D.48}$$

and g_y and g_z are defined similarly.

Next, these sources are cosine transformed in x, y, and z. To get the DFT $\hat{G}$, this triple cosine transform is scaled by a factor of 8. To illustrate, the scaled triple transform of g_x is $(-1)^\alpha G_x(\beta,\gamma)$ where

$$G_x(\beta,\gamma) = 4 \sum_{j=0}^{m_y'} \sum_{k=0}^{m_z'} \cos\left(\frac{\beta j \pi}{m_y}\right) \cos\left(\frac{\gamma k \pi}{m_z}\right) g_x(j,k) \tag{D.49}$$

and the $'$ denotes a scale factor of $1/2$ on the initial and final summands.

Finally, the triple cosine transform is divided by the sum of the three eigenvectors, as in section D.3.1. Here

$$e_x(\alpha) = -\frac{4}{\Delta x^2} \sin^2\left(\frac{\alpha\pi}{2m_x}\right) \tag{D.50}$$

and e_y and e_z are defined similarly. Thus

$$\hat{G}(\alpha,\beta,\gamma) = \frac{1 + s_\alpha G_x(\beta,\gamma) + s_\beta G_y(\alpha,\gamma) + s_\gamma G_z(\alpha,\beta)}{e_x(\alpha) + e_y(\beta) + e_z(\gamma)} \tag{D.51}$$

for $(\alpha,\beta,\gamma) \neq (0,0,0)$, $s_\alpha = (-1)^\alpha$, etc.

$\hat{G}(0,0,0)$ requires a special computation since the denominator of (D.51) is zero. But first note that the numerator must also be zero, which is the ***consistency condition***:

$$G_x(0,0) + G_y(0,0) + G_z(0,0) = -1. \tag{D.52}$$

Formula (D.52) is an excellent check of numerical accuracy. The code automatically scales the left side of (D.52) by a factor, labeled RSC, which forces (D.52) to be satisfied to machine accuracy. Then the deviation of RSC from 1 is used as a measure of the number of significant digits in the Green's function.

To compute $\hat{G}(0,0,0)$, first compute $G(m_x, m_y, m_z)$ to maximum accuracy by the asymptotic expansion (D.1). Then represent $G(m_x, m_y, m_z)$ by the inverse DFT formula

$$\begin{aligned} G(m_x,m_y,m_z) &= \frac{1}{m_x m_y m_z} \sum_{\alpha=0}^{m_x'} \sum_{\beta=0}^{m_y'} \sum_{\gamma=0}^{m_z'} \cos\alpha\pi \cos\beta\pi \cos\gamma\pi\, \hat{G}(\alpha,\beta,\gamma) \\ &= \frac{1}{m_x m_y m_z} \sum_{\alpha=0}^{m_x'} \sum_{\beta=0}^{m_y'} \sum_{\gamma=0}^{m_z'} (-1)^{\alpha+\beta+\gamma}\, \hat{G}(\alpha,\beta,\gamma) \end{aligned} \tag{D.53}$$

Solving (D.53) for $\hat{G}(0,0,0)$ gives

$$\hat{G}(0,0,0) = 8\left(m_x m_y m_z\, G(m_x,m_y,m_z) - \sum_{\alpha,\beta,\gamma}{}' (-1)^{\alpha+\beta+\gamma}\,\hat{G}(\alpha,\beta,\gamma)\right) \tag{D.54}$$

where the sum extends over all (α,β,γ), except $(0,0,0)$, as in (D.53).

D.2.4 The Downstream Green's Function

The downstream Green's function has been defined by the summation (D.4), but the actual value of this summation is always infinite, as is shown below. However it is also shown below that definition (D.4) is mathematically valid in the following sense. Interpret G_d as a linear operator on the set $\mathcal{D}_Q$ of all downstream source functions that sum to zero:

$$G_d(i,\cdot,\cdot) * \tau \;=\; \sum_{\ell=m_x}^{\infty} G(\ell - i,\cdot,\cdot) * \tau \tag{D.55}$$

for

$$\tau \in \mathcal{D}_Q \;=\; \left\{ f : \sum_{j=0}^{m_y}\sum_{k=0}^{m_z} f(i,j) = 0 \right\}. \tag{D.56}$$

If G is already known on the computational box, then only the summations outside the box need be investigated in order to compute G_d. Outside the box the asymptotic expansion (D.2) may be used to compute G. The Euler-Maclaurin formula permits the infinite sum of each asymptotic term to be computed by the corresponding infinite integral plus correction terms derived from the derivatives of the summand function at the lower boundary (downstream edge of the box). That is, write

$$G_d(i,j,k) \;=\; G_d(-1,j,k) + \sum_{\ell=m_x-i}^{m_x} G(\ell,j,k), \tag{D.57}$$

and compute

$$\begin{aligned} G_d(-1,j,k) \;&=\; \sum_{\ell=1}^{\infty} G(m_x+\ell,j,k) \\ &=\; \frac{1}{\Delta x}\int_{x_0}^{\infty} G(x,j,k)dx - \frac{1}{2}G(x_0,j,k) - \frac{\Delta x}{12}D_x G(x_0,j,k) \\ &+\; \frac{\Delta x^3}{720}D_x^3 G(x_0,j,k) - \sum_{i=3}^{\infty}\frac{B_{2i}}{(2i)!}\Delta x^{2i-1} D_x^{2i-1} G(x_0,j,k) \end{aligned} \tag{D.58}$$

where $x_0 = m_x\Delta x$ and B_{2i} is a Bernoulli number.

Applying (D.58) to the expansion (D.2) shows that the infinite part comes from the integral

$$\begin{aligned} \int_{x_0}^{N}\frac{dx}{|(x,y,z)|} \;&=\; \ln\left[N + \sqrt{N^2+y^2+z^2}\right] - \ln(x_0+r) \\ &=\; \ln(2N) - \ln(x_0+r) + O(1/N^2) \end{aligned} \tag{D.59}$$

for $N \sim \infty$ and $r = |(x_0, y, z)|$. The infinite term $\ln(x_0 + N)$ is constant for all j, k, and a constant vanishes upon convolution with a function in $\mathcal{D}_Q$. Therefore the operator formula (D.55) is well defined.

The simplest way to make G_d well defined as a function in the context of the Euler-Maclaurin expansion is to delete the infinite term from the integration (D.59). However, this is not quite enough because the goal is to compute $\hat{G}_d$ in the same way as $\hat{G}$ is computed: First compute G_d values at the boundary of the computational box, then solve a Neumann boundary value problem. For this purpose $\hat{G}_d(0,0,0)$ requires a separate computation, just as $\hat{G}(0,0,0)$ did in section D.2.3. But in this case G_d is actually only defined up to an arbitrary constant, according to (D.55) and (D.56). That is, the value of $\hat{G}_d(0,0,0)$ may be chosen arbitrarily. The simplest choice is $\hat{G}_d(0,0,0) = 0$, and this definition also has the advantage that it tends to minimize the effects of any numerical deviation from the downstream condition (D.56).

Another aspect of the Neumann boundary value approach to $\hat{G}_d$ is that four boundary planes of data, instead of three, are required due to the lack of symmetry along the x axis. The equivalent sources on the $i = 0$ and $i = m_x$ planes are

$$\begin{aligned} g_{d0}(j,k) &= -\frac{G_d(-1,j,k) - G_d(+1,j,k)}{\Delta x^2} + \delta(j,k) \\ &= \frac{G(m_x - 1, j, k) + G(m_x, j, k)}{\Delta x^2} + \delta(j,k) \end{aligned} \tag{D.60}$$

$$\begin{aligned} g_{dx}(j,k) &= -\frac{G_d(m_x + 1, j, k) - G_d(m_x - 1, j, k)}{\Delta x^2} \\ &= -\frac{G(0,j,k) + G(1,j,k)}{\Delta x^2}. \end{aligned} \tag{D.61}$$

From (D.57) the boundary plane at $j = m_y$ is

$$\begin{aligned} g_{dy}(i,k) &= -\frac{G_d(i, m_y + 1, k) - G_d(i, m_y - 1, k)}{\Delta y^2} \\ &= -\frac{G_d(-1, m_y + 1, k) - G_d(-1, m_y - 1, k)}{\Delta y^2} \\ &\quad + \sum_{\ell = m_x - i}^{m_x} g_y(\ell, k) \end{aligned} \tag{D.62}$$

and similarly for $g_{dx}(i,j)$. A scaled triple cosine transform is applied to each of planes (D.60) - (D.62), just as in section D.2.3, to get the four planes G_{d0}, G_{dx}, G_{dy}, G_{dz}. The result is

$$\hat{G}_d(\alpha,\beta,\gamma) = \frac{G_{d0}(\beta,\gamma) + s_\alpha G_{dx}(\beta,\gamma) + s_\beta G_{dy}(\alpha,\gamma) + s_\gamma G_{dz}(\alpha,\beta)}{e_x(\alpha) + e_y(\beta) + e_z(\gamma)} \tag{D.63}$$

for $(\alpha,\beta,\gamma) \neq (0,0,0)$, $\hat{G}_d(0,0,0) = 0$, $s_\alpha = (-1)^\alpha$, etc.

If G has been computed to an error of $O(1/r^7)$ by the expansion (D.2), then the infinite summation (D.4) gives a G_d error of $O(1/r^6)$ in the following way. The $1/r$

term in the expansion of G contributes

$$P(1/r) = -\ln(x_0 + r) - \frac{1}{2r} + \frac{\Delta x u}{12r^2} + \frac{\Delta x^3(9u - 15u^3)}{720r^4}. \tag{D.64}$$

where $u = x_0/r$. According to (D.43) the η_3/r^3 contribution requires three Euler-Maclaurin calculations:

$$P(u^{2i}/r^3) = \frac{1}{\Delta x}\int_{x_0}^{\infty} \frac{x^{2i}}{r^{3+2i}}dx - \frac{u^{2i}}{2r^3} - \frac{\Delta x}{12r^4}\left(2iu^{2i-1} - (3+2i)u^{2i+1}\right) \tag{D.65}$$

for $i = 0, 1, 2$. By (D.44) the η_5/r^5 contribution requires five calculations:

$$P(u^{2i}/r^5) = \frac{1}{\Delta x}\int_{x_0}^{\infty} \frac{x^{2i}}{r^{5+2i}}dx - \frac{u^{2i}}{2r^5} \tag{D.66}$$

for $i = 0, 1, 2, 3, 4$.

Using integration by parts, each of the integrals in (D.65) may be reduced to

$$\int_{x_0}^{\infty} \frac{dx}{r^3} = \frac{1}{1+u}\frac{1}{r^2}, \tag{D.67}$$

and each of the integrals in (D.66) may be reduced to

$$\int_{x_0}^{\infty} \frac{dx}{r^5} = \frac{u+2}{3(u+1)^2}\frac{1}{r^4}. \tag{D.68}$$

Thus for $(x_0, y, z) = (m_x\Delta x, j\Delta y, k\Delta z)$, $V^i(j) = \Delta y^i v^j + \Delta z^i w^j$, and $U^i(0) = \Delta x^i + \Delta y^i + \Delta z^i$, compute

$$\begin{aligned}
G_d(-1, j, k) = -\frac{\Delta x \Delta y \Delta z}{4\pi}\Big[& P(1/r) + \left(\frac{1}{8}U^2(0) - \frac{3}{4}V^2(2) + \frac{5}{8}V^2(4)\right)P(1/r^3) \\
- & \frac{3}{4}\Delta x^2 P(u^2/r^3) + \frac{5}{8}\Delta x^2 P(u^4/r^3) + \left(\frac{3}{16}U^4(0) - \frac{65}{16}V^4(2) + \frac{105}{8}V^4(4)\right. \\
- & \frac{189}{16}V^4(6) + \left(\frac{3}{128}U^2(0) - \frac{15}{32}V^2(2) + \frac{35}{64}V^2(4)\right)U^2(0) \\
+ & \left(\frac{105}{32}V^2(2) - \frac{315}{32}V^2(4)\right)V^2(2) + \left.\frac{1155}{128}V^2(4)V^2(4)\right)P(1/r^5) \\
+ & \left(-\frac{65}{16}\Delta x^2 - \frac{15}{32}U^2(0) + \frac{105}{16}V^2(2) - \frac{315}{32}V^2(4)\right)\Delta x^2 P(u^2/r^5) \\
+ & \left(\frac{35}{64}U^2(0) + \frac{525}{32}\Delta x^2 - \frac{315}{32}V^2(2) + \frac{1155}{64}V^2(4)\right)\Delta x^2 P(u^4/r^5) \\
- & \frac{693}{32}\Delta x^4 P(u^6/r^5) + \frac{1155}{128}\Delta x^4 P(u^8/r^5)\Big].
\end{aligned} \tag{D.69}$$

D.3 THEORY OF THE CONVOLUTION ALGORITHM

D.3.1 FFT Dirichlet and Neumann Poisson Solvers

FFT Poisson solver methods are described for a rectangular domain (box) in a Cartesian grid of arbitrary dimension. At each face the boundary conditions may be chosen

independently to be Dirichlet or Neumann. Neumann boundary conditions are of the central difference kind, centered on a face.

Additional boundary options include periodic boundary conditions and also Dirichlet or Neumann boundary conditions of the simple forward or backward difference kind, centered on a half grid line just inside a face.

The methods employ FFT sine and cosine transforms and shifted sine and cosine transforms. Optionally, tridiagonal solvers, which are faster, may replace the "last" transform.

These transforms are derived as eigenvectors of the matrices that describe the problems. For example, a 1D Poisson problem $D^2\phi = \rho$ may be written as a matrix equation $A\mathbf{x} = \mathbf{b}$ and solved by computing $\mathbf{x} = E(\Lambda^{-1}(E^{-1}\mathbf{b}))$, where E is the eigenvector matrix and Λ is the diagonal eigenvalue matrix. In general such a procedure is extremely inefficient, but the availability of analytical formulas for Λ and E and of FFT techniques for the matrix-vector multiplies makes the method very attractive. Swarztrauber[94] also presents much of the following material.

Derivation of the Matrix Equations

In a 1D Dirichlet problem $D^2\phi = \rho$ the endpoint values $\phi(0)$ and $\phi(m)$ are specified. Thus the problem may be rewritten as the matrix equation

$$A\mathbf{x} = \mathbf{b} \tag{D.70}$$

where A and b are given by

$$\begin{aligned} -2x_1 + x_2 &= \Delta x^2\rho(1) - \phi(0) \\ &\cdots \\ x_{j-1} - 2x_j + x_{j+1} &= \Delta x^2\rho(j) \\ &\cdots \\ x_{m-2} - 2x_{m-1} &= \Delta x^2\rho(m-1) - \phi(m). \end{aligned} \tag{D.71}$$

Next let $D^2\phi = \rho$ represent a central difference Neumann problem with endpoint specifications $d_b = (\phi(-1) - \phi(1))/2\Delta x$ and $d_e = (\phi(m+1) - \phi(m-1))/2\Delta x$. Then the matrix equation is given by

$$\begin{aligned} -2x_0 + 2x_1 &= \Delta x^2\rho(0) - 2\Delta x d_b \\ &\cdots \\ x_{j-1} - 2x_j + x_{j+1} &= \Delta x^2\rho(j) \\ &\cdots \\ 2x_{m-1} - 2x_m &= \Delta x^2\rho(m) - 2\Delta x d_e. \end{aligned} \tag{D.72}$$

A Dirichlet condition on the left with a Neumann condition on the right gives

$$\begin{aligned} -2x_1 + x_2 &= \Delta x^2\rho(1) - \phi(0) \\ &\cdots \\ x_{j-1} - 2x_j + x_{j+1} &= \Delta x^2\rho(j) \\ &\cdots \\ 2x_{m-1} - 2x_m &= \Delta x^2\rho(m) - 2\Delta x d_e. \end{aligned} \tag{D.73}$$

A Dirichlet condition on the right with a Neumann on the left gives

$$\begin{aligned} -2x_0 + 2x_1 &= \Delta x^2 \rho(0) - 2\Delta x d_b \\ &\cdots \\ x_{j-1} - 2x_j + x_{j+1} &= \Delta x^2 \rho(j) \\ &\cdots \\ x_{m-2} - 2x_{m-1} &= \Delta x^2 \rho(m-1) - \phi(m). \end{aligned} \tag{D.74}$$

For the periodic boundary condition $\phi(m+j) = \phi(j)$, the result is

$$\begin{aligned} -2x_0 + x_1 \qquad + x_{m-1} &= \Delta x^2 \rho(0) \\ &\cdots \\ x_{j-1} - 2x_j + x_{j+1} &= \Delta x^2 \rho(j) \\ &\cdots \\ x_0 \qquad + x_{m-2} - 2x_{m-1} &= \Delta x^2 \rho(m-1). \end{aligned} \tag{D.75}$$

For Dirichlet half grid line boundary conditions, $a_b = (\phi(0) + \phi(1))/2$ and $a_e = (\phi(m-1) + \phi(m))/2$ are specified. This gives

$$\begin{aligned} -3x_1 + x_2 &= \Delta x^2 \rho(1) - 2a_b \\ &\cdots \\ x_{j-1} - 2x_j + x_{j+1} &= \Delta x^2 \rho(j) \\ &\cdots \\ x_{m-2} - 3x_{m-1} &= \Delta x^2 \rho(m-1) - 2a_e. \end{aligned} \tag{D.76}$$

For Neumann half grid line boundary conditions, $d_b = (\phi(0) - \phi(1))/\Delta x$ and $d_e = (\phi(m) - \phi(m-1))/\Delta x$ are specified. This gives

$$\begin{aligned} -x_1 + x_2 &= \Delta x^2 \rho(0) - \Delta x d_b \\ &\cdots \\ x_{j-1} - 2x_j + x_{j+1} &= \Delta x^2 \rho(j) \\ &\cdots \\ x_{m-2} - x_{m-1} &= \Delta x^2 \rho(m-1) - \Delta x d_e. \end{aligned} \tag{D.77}$$

Of course, mixed half grid line Dirichlet-Neumann problems are also possible by starting out as in (D.76) and ending as in (D.77), or vice versa.

If the Poisson problem is a 2D problem, let A_x be the 1D matrix for the x direction and A_y be the 1D matrix for the y direction. Let I_x and I_y be the corresponding identity matrices, and let $\otimes$ denote the tensor product. Then the matrix equation becomes

$$\left(\frac{A_x \otimes I_y}{\Delta x^2} + \frac{I_x \otimes A_y}{\Delta y^2} \right) \mathbf{x} = \mathbf{b}. \tag{D.78}$$

(The tensor product of two matrices $A_{n \times m}$ and $B_{o \times p}$ is a linear operator on matrices of order $m \times p$ that gives $n \times o$ matrices. It may be defined by

$$(A \otimes B)\mathbf{x} = \sum_i A_{ki} y_{i\ell} \quad \text{for} \quad y_{i\ell} = \left(\sum_j B_{\ell j} x_{ij}^t \right)^t$$

or, equivalently,

$$(A \otimes B)\mathbf{x} = \left(\sum_j B_{\ell j} z_{kj}^t \right)^t \quad \text{for} \quad z_{kj} = \sum_i A_{ki} x_{ij}.$$

That is, first operate on the rows by B, then on the columns of the result by A, or reverse the order to operate first on the columns by A, then on the rows by B.) If, for example, A_x is Neumann on the left and Dirichlet on the right with A_y full Dirichlet, the $\mathbf{b}$ in (D.78) is given by the computer operations

$$\begin{aligned}
b_{ij} &:= \rho(i,j) && \text{for } 0 \le i < m_x,\ 0 < j < m_y \\
b_{0j} &:= b_{0j} - \frac{2d_b(j)}{\Delta x} && \text{for } 0 < j < m_y \\
b_{m_x-1,j} &:= b_{m_x-1,j} - \frac{\phi(m_x,j)}{\Delta x^2} && \text{for } 0 < j < m_y \\
b_{i1} &:= b_{i1} - \frac{\phi(i,0)}{\Delta y^2} && \text{for } 0 \le i < m_x \\
b_{i,m_y-1} &:= b_{i,m_y-1} - \frac{\phi(i,m_y)}{\Delta y^2} && \text{for } 0 \le i < m_x.
\end{aligned} \tag{D.79}$$

Both (D.78) and (D.79) generalize directly to higher dimensions, with the 3D equation being

$$\left(\frac{A_x \otimes I_y \otimes I_z}{\Delta x^2} + \frac{I_x \otimes A_y \otimes I_z}{\Delta y^2} + \frac{I_x \otimes I_y \otimes A_z}{\Delta z^2} \right) \mathbf{x} = \mathbf{b}. \tag{D.80}$$

Eigenvectors and Eigenvalues

The eigenvector-eigenvalue solution to equation (D.70) is simply

$$\mathbf{x} = A^{-1}\mathbf{b} = (E\Lambda E^{-1})^{-1}\mathbf{b} = E(\Lambda^{-1}(E^{-1}\mathbf{b})) \tag{D.81}$$

where $E = (e_{jk})$ is the eigenvector matrix of A and $\Lambda = (\lambda_k)$ is the diagonal eigenvalue matrix. This generalizes directly to the multi-dimensional tensor product formulation as follows. Applying the tensor product properties

$$\begin{aligned}
(A \otimes B)[(C + C') \otimes (D + D')] &= (AC + AC') \otimes (BD + BD') \\
[(A + A') \otimes (B + B')](C \otimes D) &= (AC + A'C) \otimes (BD + B'D)
\end{aligned}$$

to (D.78), for example, gives

$$\mathbf{x} = (E_x \otimes E_y) \left(\frac{\Lambda_x \otimes I_y}{\Delta x^2} + \frac{I_x \otimes \Lambda_y}{\Delta y^2} \right)^{-1} \left(E_x^{-1} \otimes E_y^{-1} \right) \mathbf{b}. \tag{D.82}$$

The corresponding 3D solution is

$$\mathbf{x} = (E_x \otimes E_y \otimes E_z)\, \mathbf{y} \tag{D.83}$$

where

$$\mathbf{y} = \left(\frac{\Lambda_x \otimes I_y \otimes I_z}{\Delta x^2} + \frac{I_x \otimes \Lambda_y \otimes I_z}{\Delta y^2} + \frac{I_x \otimes I_y \otimes \Lambda_z}{\Delta z^2} \right)^{-1} \mathbf{b}^{xyz}$$

and

$$\mathbf{b}^{xyz} = \left(E_x^{-1} \otimes E_y^{-1} \otimes E_z^{-1} \right) \mathbf{b}.$$

Another way to write (D.83) is

$$\left(\frac{\lambda_x(\alpha)}{\Delta x^2} + \frac{\lambda_y(\beta)}{\Delta y^2} + \frac{\lambda_z(\gamma)}{\Delta z^2} \right) \mathbf{x}^{xyz}(\alpha, \beta, \gamma) = \mathbf{b}^{xyz}(\alpha, \beta, \gamma). \tag{D.84}$$

The three plane representation formula (D.37) for the Green's function is derived from (D.84).

Now the task is to find the eigenvalues and eigenvectors for the matrices (D.71) - (D.77). These are given in table D.1.

There are certain useful relations between some of the eigenvector matrices of table D.1. Let S be the operator that reverses the order of the rows when multiplied on the left, and let T negate the even numbered columns when multiplied on the right. Then

$$\begin{aligned}
E_{D_h D_h}(m) &= E_{DN}^t(m-1) \\
E_{N_h N_h}(m) &= E_{ND}^t(m-1) \\
E_P(m) &= E_{NN}(m/2) + E_{DD}(m/2)i \\
E_{DN} &= S\, E_{ND}\, T \\
E_{D_h D_h} &= T\, E_{N_h N_h}\, S \\
E_{D_h N_h} &= S\, E_{N_h D_h}\, T.
\end{aligned} \tag{D.85}$$

Table D.1 may be verified by substituting into the appropriate matrix equation, (D.71) - (D.77), and using trigonometric identities. For example, in the half grid line Neumann case (D.77) let $n = m - 1$ and $\theta = k\pi/2n$. Then

$$\begin{aligned}
(A\mathbf{e}_k)_1 &= -\cos\theta + \cos 3\theta \\
&= -\cos\theta + \cos\theta\cos 2\theta - \sin\theta \sin 2\theta \\
&= \cos\theta[-1 + (1 - 2\sin^2\theta) - 2\sin^2\theta] \\
&= -4\sin^2\theta\cos\theta \\
(A\mathbf{e}_k)_j &= \cos(2j-3)\theta - 2\cos(2j-1)\theta + \cos(2j+1)\theta \\
&= (\cos 2\theta - 2 + \cos 2\theta)\cos(2j-1)\theta \\
&= -4\sin^2(\theta)\cos(2j-1)\theta \\
(A\mathbf{e}_k)_n &= \cos(2n-3)\theta - \cos(2n-1)\theta \\
&= (-1)^k(\cos 3\theta - \cos\theta) \\
&= -4\sin^2\theta\cos(2n-1)\theta
\end{aligned}$$

There is one case in which the solution formula (D.83) fails. Namely, a zero sum-of-eigenvalues cannot be inverted. According to table D.1 this occurs for $\alpha = \beta = \gamma = 0$

Problem*	Eigenvectors E	Eigenvalues Λ	Indices
DD	$\sin\left(\frac{jk\pi}{m}\right)$	$-4\sin^2\left(\frac{k\pi}{2m}\right)$	$\begin{cases} 1 \le j \le m-1 \\ 1 \le k \le m-1 \end{cases}$
NN	$\cos\left(\frac{jk\pi}{m}\right)$	$-4\sin^2\left(\frac{k\pi}{2m}\right)$	$\begin{cases} 0 \le j \le m \\ 0 \le k \le m \end{cases}$
DN	$\sin\left(\frac{j(k-\frac{1}{2})\pi}{m}\right)$	$-4\sin^2\left(\frac{(k-\frac{1}{2})\pi}{2m}\right)$	$\begin{cases} 1 \le j \le m \\ 1 \le k \le m \end{cases}$
ND	$\cos\left(\frac{j(k-\frac{1}{2})\pi}{m}\right)$	$-4\sin^2\left(\frac{(k-\frac{1}{2})\pi}{2m}\right)$	$\begin{cases} 0 \le j \le m-1 \\ 1 \le k \le m \end{cases}$
P	$\exp\left\{\frac{2jk\pi i}{m}\right\}$	$-4\sin^2\left(\frac{k\pi}{m}\right)$	$\begin{cases} 0 \le j \le m-1 \\ 0 \le k \le m-1 \end{cases}$ **Key***
D_hD_h	$\sin\left(\frac{(j-\frac{1}{2})k\pi}{m-1}\right)$	$-4\sin^2\left(\frac{k\pi}{2(m-1)}\right)$	$\begin{cases} 1 \le j \le m-1 \\ 1 \le k \le m-1 \end{cases}$
N_hN_h	$\cos\left(\frac{(j-\frac{1}{2})k\pi}{m-1}\right)$	$-4\sin^2\left(\frac{k\pi}{2(m-1)}\right)$	$\begin{cases} 1 \le j \le m-1 \\ 0 \le k \le m-2 \end{cases}$
D_hN_h	$\sin\left(\frac{(j-\frac{1}{2})(k-\frac{1}{2})\pi}{m-1}\right)$	$-4\sin^2\left(\frac{(k-\frac{1}{2})\pi}{2(m-1)}\right)$	$\begin{cases} 1 \le j \le m-1 \\ 1 \le k \le m-1 \end{cases}$
N_hD_h	$\cos\left(\frac{(j-\frac{1}{2})(k-\frac{1}{2})\pi}{m-1}\right)$	$-4\sin^2\left(\frac{(k-\frac{1}{2})\pi}{2(m-1)}\right)$	$\begin{cases} 1 \le j \le m-1 \\ 1 \le k \le m-1 \end{cases}$

D = standard Dirichlet N = central difference Neumann
D_h = half grid line Dirichlet N_h = half grid line Neumann
P = periodic

Table D.1: Eigenvectors and Eigenvalues

when every face has a central difference Neumann boundary condition. In this case the solution $\mathbf{x}$ is determined only up to an additive constant, and an additional equation is needed to determine this constant. In addition (D.84) shows that the *consistency condition*

$$\mathbf{b}^{xyz}(0,0,0) = 0 \tag{D.86}$$

must hold. Again, the three plane Green's function formula (D.37) is an instance.

The Transforms

To implement FFT algorithms for the eigenvector matrices of table D.1, several modifications are useful. First note that if F is any diagonal matrix, then $A = E\Lambda E^{-1} = (EF)\Lambda(EF)^{-1}$. That is, the eigenvectors may be scaled anyway you please. For example, if $F_{kk} = 1$ except $F_{11} = F_{mm} = \frac{1}{2}$, then the cosine transform E of table D.1 may be replaced by the standard cosine transform EF.

But note that in (D.81) - (D.83) it is the inverse of an eigenvector matrix that is first applied to the data vector $\mathbf{b}$. Therefore the scale factors F are chosen so that $(EF)^{-1}$ is a standard forward transform. These transforms and their inverses are listed in table D.2.

The T_{DN} and T_{ND} transforms, or their inverses $T_{D_hD_h}$ and $T_{N_hN_h}$, are referred to as the *shifted sine* and *shifted cosine* transforms, respectively. The $T_{D_hN_h}$ and $T_{N_hD_h}$ transforms are the *dual shifted sine* and *dual shifted cosine* transforms.

A straightforward way to derive the eigenvector matrix inverses of table D.2 is to compute $S = (s_k)$ for $s_k = |\mathbf{e}_k|^2$ and $\mathbf{e}_k$ = the kth eigenvector. Then $E^{-1} = S^{-1}E^t$. Another way is to reduce the problem to standard DFT inversion by using symmetric or antisymmetric data. For example, to invert T_{N_h,N_h} set $b_{2n+1-j} = b_j$ for $j = 1...n$ since $\cos(k(2n+1-j-\frac{1}{2})\pi i/n) = \cos(k(j-\frac{1}{2})\pi i/n)$. Also make the data periodic of period $2n$ so that $b_0 = b_{2n}$. Then

$$\begin{aligned}
\hat{b}_k &= \left(T_{N_hN_h}\mathbf{b}\right)(k) = \sum_{j=1}^{n} \cos\left(k(j-\frac{1}{2})\frac{\pi}{n}\right) b_j \\
&= \frac{1}{2}\sum_{j=0}^{2n-1} \exp\left\{k(j-\frac{1}{2})\frac{\pi i}{n}\right\} b_j \\
&= \frac{1}{2}\exp\{-k\pi i/2n\}\sum_{j=0}^{2n-1} \exp\left\{kj\frac{\pi i}{n}\right\} b_j.
\end{aligned} \tag{D.87}$$

Now invert the DFT and use $\hat{b}_k = \hat{b}_{2n-k}$ and $\hat{b}_n = 0$ to get

$$\begin{aligned}
b_j &= \frac{1}{2n}\sum_{k=0}^{2n-1} \exp\left\{-jk\frac{\pi i}{n}\right\} 2\exp\{k\pi i/2n\}\hat{b}_k \\
&= \frac{1}{n}\sum_{k=0}^{2n-1} \exp\left\{-(j-\frac{1}{2})k\frac{\pi i}{n}\right\}\hat{b}_k \\
&= \frac{2}{n}\left[\frac{\hat{b}_0}{2} + \sum_{k=1}^{n-1} \cos\left((j-\frac{1}{2})k\frac{\pi}{n}\right)\hat{b}_k\right] = \frac{2}{n}\left(T_{ND}\hat{\mathbf{b}}\right)(j).
\end{aligned}$$

Forward Transform*	Inverse Transform
$(T_{DD}\mathbf{b})(k) = \sum_{j=1}^{m-1} \sin\left(\frac{kj\pi}{m}\right) b_j$	$T_{DD}^{-1} = \frac{2}{m} T_{DD}$
$(T_{NN}\mathbf{b})(k) = \frac{b_0}{2} + \sum_{j=1}^{m-1} \cos\left(\frac{kj\pi}{m}\right) b_j + (-1)^k \frac{b_m}{2}$	$T_{NN}^{-1} = \frac{2}{m} T_{NN}$
$(T_{DN}\mathbf{b})(k) = \sum_{j=1}^{m-1} \sin\left(\frac{(k-\frac{1}{2})j\pi}{m}\right) b_j + (-1)^{k-1} \frac{b_m}{2}$	$T_{DN}^{-1} = \frac{2}{m} T_{D_h D_h}(m+1)$
$(T_{ND}\mathbf{b})(k) = \frac{b_0}{2} + \sum_{j=1}^{m-1} \cos\left(\frac{(k-\frac{1}{2})j\pi}{m}\right) b_j$	$T_{ND}^{-1} = \frac{2}{m} T_{N_h N_h}(m+1)$
$(T_P\mathbf{b})(k) = \sum_{j=0}^{m-1} \exp\left(\frac{2kj\pi i}{m}\right) b_j$	$T_P^{-1} = \frac{1}{m} T_P$
$(T_{D_h D_h}\mathbf{b})(k) = \sum_{j=1}^{m-1} \sin\left(\frac{k(j-\frac{1}{2})\pi}{m-1}\right) b_j$	$T_{D_h D_h}^{-1} = \frac{2}{m-1} T_{DN}(m-1)$
$(T_{N_h N_h}\mathbf{b})(k) = \sum_{j=1}^{m-1} \cos\left(\frac{k(j-\frac{1}{2})\pi}{m-1}\right) b_j$	$T_{N_h N_h}^{-1} = \frac{2}{m-1} T_{ND}(m-1)$
$(T_{D_h N_h}\mathbf{b})(k) = \sum_{j=1}^{m-1} \sin\left(\frac{(k-\frac{1}{2})(j-\frac{1}{2})\pi}{m-1}\right) b_j$	$T_{D_h N_h}^{-1} = \frac{2}{m-1} T_{D_h N_h}$
$(T_{N_h D_h}\mathbf{b})(k) = \sum_{j=1}^{m-1} \cos\left(\frac{(k-\frac{1}{2})(j-\frac{1}{2})\pi}{m-1}\right) b_j$	$T_{N_h D_h}^{-1} = \frac{2}{m-1} T_{N_h D_h}$

Key*

D = standard Dirichlet N = central difference Neumann
D_h = half grid line Dirichlet N_h = half grid line Neumann
P = periodic

Table D.2: Transforms and Their Inverses

Tridiagonal Solvers

Instead of doing three transforms in (D.83), forwards then backwards, it is computationally more efficient to do two forward transformations, a tridiagonal solver, then two inverse transformations. For example, if the eigenvector decomposition is used only in the x and z directions, then the following equation is obtained instead.

$$\mathbf{x} = (E_x \otimes I_y \otimes E_z)\,\mathbf{y} \tag{D.88}$$

where

$$\mathbf{y} = \left(\frac{\Lambda_x \otimes I_y \otimes I_z}{\Delta x^2} + \frac{I_x \otimes A_y \otimes I_z}{\Delta y^2} + \frac{I_x \otimes I_y \otimes \Lambda_z}{\Delta z^2}\right)^{-1} \mathbf{b}^{xz}$$

and

$$\mathbf{b}^{xz} = \left(E_x^{-1} \otimes I_y \otimes E_z^{-1}\right)\mathbf{b}.$$

Formula (D.88) is equivalent to solving the following tridiagonal equation for every α and γ.

$$\left[A_y + \left(\frac{\lambda_x(\alpha)}{\Delta x^2} + \frac{\lambda_z(\gamma)}{\Delta z^2}\right) I_y\right] \mathbf{x}^{xz}(\alpha,\cdot,\gamma) = \mathbf{b}^{xz}(\alpha,\cdot,\gamma) \tag{D.89}$$

The n-dimensional generalization of (D.88) and (D.89) is obtained by specifying an order for the n transforms in the generalization of (D.83), then simply omitting the last one.

A general purpose tridiagonal solver could be used, but the (almost) equal and constant sub and super diagonals and the (almost) constant main diagonal permit a more efficient implementation. The code uses a modification of the LINPACK symmetric matrix algorithm, with vectorization in the x direction, unrolled loops in y, and an outer loop in z. This way the FFT in x vectorizes over yz planes and the FFT in z vectorizes over xy planes for the ordering in an xyz FORTRAN array.

The idea behind the algorithm is to do Gaussian elimination from the top and bottom simultaneously until the process meets in the middle. Then do back substitutions from the middle back to the top and bottom simultaneously. If the number of rows is even, there is a middle 2 by 2 block, which is solved directly. For an odd number of rows, both off diagonal values of the center row are eliminated simultaneously to get the middle value.

To minimize the number of operations, especially the number of divisions, the following technique is used for a problem with constant diagonal and off diagonal. Define a = main diagonal value, c = off diagonal value, a_j = jth row diagonal value after the $j-1$st Gaussian step, b_j = jth row right hand side , $d_j = c/a_j$, $d_0 = a/c$, and $c_1 = 1/c$. Only the values d_0, c_1, b_j, and d_j are actually computed and stored. Then in the forward substitution

$$a_1 = a \quad \text{and} \quad a_{j+1} = a - \frac{c^2}{a_j}$$

so

$$d_{j+1} = \frac{a_j c}{a a_j - c^2} = \frac{1}{d_0 - d_j},$$

$$
\begin{aligned}
b_{j+1} &:= b_{j+1} - d_j b_j, \\
b_{n-j} &:= b_{n-j} - d_j b_{n+1-j}.
\end{aligned}
$$

In the back substitution

$$
\begin{aligned}
b_j &:= (b_j - c b_{j+1})/a_j = d_j(c_1 b_j - b_{j+1}) \\
b_{n-j} &:= d_j(c_1 b_{n-j} - b_{n-1-j}).
\end{aligned}
$$

In the case of a mixed Dirichlet-Neumann problem the diagonal values a_j, hence d_j, are different for the top and bottom eliminations. Also the first step of the forward substitution and the last step of the backward substitution must be specially coded.

D.3.2 Transform Algorithms

Several common transform algorithms are used in the code and also several unusual or specialized algorithms. All the algorithms consist of reductions to the standard complex FFT. In most cases there is an intermediate reduction to real FFT's. The simplest and most efficient method of coding real FFT's was to do a complex FFT on a pair of real sequences, although a single-sequence real transform code would have been simpler to use. Therefore most of the algorithms are actually applied to a pair of sequences. All the algorithms are implemented for a matrix-type data structure so that the transforms are performed in one direction with vectorization, or processing in parallel, in the other direction.

First, let's review the standard real, sine, and cosine algorithms. A basic reference is [95]. Let $b_0, ..., b_{n-1}$ and $c_0, ..., c_{n-1}$ be a pair of real sequences and set $\mathbf{a} = \mathbf{b} + i\mathbf{c}$. Let the DFT (Discrete Fourier Transform) be defined by

$$
\hat{a}(k) = \mathrm{DFT}(\mathbf{a})(k) = \sum_{j=0}^{n-1} \exp\left\{kj\frac{2\pi i}{n}\right\} a_j. \tag{D.90}
$$

Then the paired real algorithm is

$$
\begin{aligned}
\hat{b}(k) &= \frac{1}{2}[\mathrm{Re}(\hat{a}(k)) + \mathrm{Re}(\hat{a}(n-k))] + \frac{i}{2}[\mathrm{Im}(\hat{a}(k)) - \mathrm{Im}(\hat{a}(n-k))] \\
\hat{c}(k) &= \frac{1}{2}[\mathrm{Im}(\hat{a}(k)) + \mathrm{Im}(\hat{a}(n-k))] + \frac{i}{2}[-\mathrm{Re}(\hat{a}(k)) + \mathrm{Re}(\hat{a}(n-k))]
\end{aligned} \tag{D.91}
$$

for $k = 1, ..., n/2$, with $\hat{b}(0) = \mathrm{Re}(\hat{a}(0))$, and $\hat{c}(0) = \mathrm{Im}(\hat{a}(0))$. Implicitly $\hat{b}(n-k) = \bar{\hat{b}}(k)$ and $\hat{c}(n-k) = \bar{\hat{c}}(k)$.

The sine transform algorithm is taken from Temperton [96]. The idea is to form a certain real sequence, apply a real transform, then extract the sine transform from the resulting data. The real sequence is

$$
d_j = \sin\left(j\frac{\pi}{n}\right)(b_j + b_{n-j}) + \frac{1}{2}(b_j - b_{n-j})
$$

for $1 \le j \le n-1$ with $d_0 = 0$. Let $\tilde{b} = T_{DD}\mathbf{b}$. Then

$$\begin{aligned} \tilde{b}(1) &= \frac{1}{2}\mathrm{Re}(\hat{d}(0)) \\ \tilde{b}(2k+1) - \tilde{b}(2k-1) &= \mathrm{Re}(\hat{d}(k)) \\ \tilde{b}(2k) &= \mathrm{Im}(\hat{d}(k)) \end{aligned} \tag{D.92}$$

for $k = 1, ..., n/2$.

The cosine transform is a close variation on (D.92). Let

$$c_j = \sin\left(j\frac{\pi}{n}\right)(b_j - b_{n-j}) + \frac{1}{2}(b_j + b_{n-j})$$

for $0 \le j \le n-1$. Let $\tilde{b} = T_{NN}\mathbf{b}$. Then

$$\begin{aligned} \tilde{b}(1) &= \frac{1}{2}(b_0 - b_n) + \sum_{j=1}^{n-1} \cos(j\pi/n) b_j \\ \tilde{b}(2k+1) - \tilde{b}(2k-1) &= \mathrm{Im}(\hat{c}(k)) \\ \tilde{b}(2k) &= \mathrm{Re}(\hat{c}(k)) \end{aligned} \tag{D.93}$$

for $k = 1, ..., n/2$.

The three transform algorithms (D.91) - (D.93) may be combined to transform a single, real, zero-extended sequence. This is a sequence $\mathbf{x} = b_0, ..., b_n, 0, ..., 0$ of length $2n$, which arises naturally in FFT convolutions. First double b_0 and b_n so that

$$\hat{x}(k) = (T_{NN}\mathbf{b})(k) + i(T_{DD}\mathbf{b})(k).$$

Next compute the real sequences d_j and c_j as in (D.92) and (D.93). Now pair up c_j and d_j to compute $\hat{c}$ and $\hat{d}$ by (D.91). Then the sine and cosine transforms, hence $\hat{x}$, are given by (D.92) and (D.93).

Note that a shifted cosine algorithm could be implemented by the symmetric data argument (D.87). Similar algorithms exist for the sine, cosine, and shifted sine. However in all these cases the DFT is of length $2n$, twice as long as it need be. A suitable shifted cosine algorithm is actually somewhat easier to derive than the sine and cosine algorithms cited above. To compute $\tilde{b} = T_{N_h N_h}\mathbf{b}$ first define $c_j = b_{2j}$. Then note that $c_{n-j} = b_{2n+1-(2j+1)} = b_{2j+1}$ and that $b_0 = b_1 = b_{2n}$ by the inverse T_{ND} formula in table D.2. That is, $\mathbf{c}$ may be computed as

$$\begin{aligned} c_0 &= b_1 \\ c_j &= b_{2j} \quad \text{for } j = 1, ..., n/2 \\ c_{n-j} &= b_{2j+1} \text{ for } j = 1, ..., (n-1)/2. \end{aligned} \tag{D.94}$$

Now write

$$c_j = \frac{1}{n}\tilde{b}(0) + \frac{2}{n}\sum_{k=1}^{n-1} \cos\left((2j - \frac{1}{2})\frac{k\pi}{n}\right)\tilde{b}(k)$$

$$= \frac{1}{n}\tilde{b}(0) + \frac{1}{n}\sum_{k=1}^{n-1}\left[\exp\left\{-(2j-\frac{1}{2})\frac{k\pi i}{n}\right\}\tilde{b}(k) + \exp\left\{(2j-\frac{1}{2})\frac{(n-k)\pi i}{n}\right\}\tilde{b}(n-k)\right]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1}\exp\left\{-jk\frac{2\pi i}{n}\right\}z_k$$

where

$$z_0 = \tilde{b}(0) \quad \text{and} \quad z_k = \exp\left\{\frac{k\pi i}{2n}\right\}\left(\tilde{b}(k) - i\tilde{b}(n-k)\right) \tag{D.95}$$

for $k = 1, .., n-1$. Thus to compute $\tilde{b}$, first apply a real transform to $\mathbf{c}$ to get $\mathbf{z}$. Then scale $\mathbf{z}$ by $\exp\{-k\pi i/2n\}$ for $1 \le k \le n/2$ and take the real part to get $\tilde{b}(k)$ and the imaginary part to get $-\tilde{b}(n-k)$. To compute T_{ND} simply reverse this procedure: Compute $\mathbf{z}$ by (D.95) for $k = 1, ..., n/2$ and apply the inverse of the real transform (D.91) to get $\mathbf{c}$, hence $\mathbf{b}$ by (D.94). An alternative algorithm is given by Swarztrauber[94].

The shifted sine algorithm is completely analogous to the shifted cosine algorithm. The difference is only that in (D.95), $\tilde{b}(0)$ is replaced by $(-1)^{k-1}\tilde{b}(n)$ and $\tilde{b}(k) - i\tilde{b}(n-k)$ is replaced by $(-\tilde{b}(k) - i\tilde{b}(n-k))/i = -\tilde{b}(n-k) + i\tilde{b}(k)$. Equivalently, the relation in (D.85) could be used to compute the shifted sine transform from the shifted cosine transform, or vice versa.

Efficient dual shifted sine and cosine algorithms may be derived by natural modifications of the shifted sine and cosine algorithms. For the dual shifted cosine transform $T_{N_hD_h}$, again define $\mathbf{c}$ as in (D.94) and modify (D.95) to get

$$c_j = \frac{2}{n}\sum_{k=1}^{n}\cos\left((2j-\frac{1}{2})\frac{(k-\frac{1}{2})\pi}{n}\right)\tilde{b}(k)$$

$$= \exp\left\{\frac{j\pi i}{n}\right\}\frac{1}{n}\sum_{k=0}^{n-1}\exp\left\{-jk\frac{2\pi i}{n}\right\}z_k$$

where

$$z_k = \exp\left\{(k-\frac{1}{2})\frac{\pi i}{2n}\right\}\left(\tilde{b}(k) - i\tilde{b}(n+1-k)\right) \tag{D.96}$$

for $1 \le k \le n$ with $z_0 = z_n$. The problem now is that a transform must be applied to the complex sequence $\exp\{-j\pi i/n\}c_j$ to solve for $\mathbf{z}$. That is, the paired real transform (D.91) can no longer be used, resulting in twice as much work as necessary. The solution is to apply the complex FFT directly to a pair of problems in analogy with algorithm (D.91). Thus set

$$c_j = \exp\left\{\frac{-j\pi i}{n}\right\}\left(c_j^1 + ic_j^2\right)$$

where c^1 and c^2 are defined by (D.94) from b^1 and b^2. Then transform $\mathbf{c}$ to get $\mathbf{z}^1 + i\mathbf{z}^2$ so that

$$\tilde{b}^1(k) = \frac{1}{2}\left(\mathrm{Re}(y_k) - \mathrm{Im}(y_{n+1-k})\right)$$

(D.97)

$$\tilde{b}^2(k) = \frac{1}{2}\left(\mathrm{Im}(y_k) + \mathrm{Re}(y_{n+1-k})\right)$$

for

$$\begin{aligned} y_k &= \exp\left\{-(k-\frac{1}{2})\frac{\pi i}{2n}\right\}(z_k^1 + iz_k^2) \\ &= \left(\tilde{b}^1(k) + \tilde{b}^2(n+1-k)\right) + i\left(-\tilde{b}^1(n+1-k) + \tilde{b}^2(k)\right). \end{aligned}$$

Just as with the shifted sine transform, there is a dual shifted sine transform analogous to the dual shifted cosine transform. The resulting formulas that replace (D.97) are

$$\begin{aligned} \tilde{b}^1(k) &= \frac{1}{2}\left(\mathrm{Im}(y_k) - \mathrm{Re}(y_{n+1-k})\right) \\ \tilde{b}^2(k) &= -\frac{1}{2}\left(\mathrm{Re}(y_k) + \mathrm{Im}(y_{n+1-k})\right) \end{aligned} \qquad \text{(D.98)}$$

for

$$y_k = -\left(\tilde{b}^1(n+1-k) + \tilde{b}^2(k)\right) + i\left(\tilde{b}^1(k) - \tilde{b}^2(n+1-k)\right).$$

Again, the relation in (D.85) may be used instead.

One more transform is useful for FFT convolutions with symmetric data. Let $\mathbf{x} = b_0, ..., b_{n-1}, b_n, b_{n-1}, ..., b_0, 0, ..., 0$ be a symmetric, zero-extended sequence of length $4n$. An efficient algorithm is derived for the DFT as follows.

$$\begin{aligned} \hat{x}(k) &= \exp\left\{kn\frac{2\pi i}{4n}\right\} b_n + \sum_{j=0}^{n-1}\left[\exp\left\{kj\frac{2\pi i}{4n}\right\} + \exp\left\{k(2n-j)\frac{2\pi i}{4n}\right\}\right] b_j \\ &= e^{k\pi i/2} b_n + \sum_{j=0}^{n-1}\left[\exp\left\{kj\frac{\pi i}{2n}\right\} + (-1)^k \exp\left\{-kj\frac{\pi i}{2n}\right\}\right] b_j \\ &= \begin{cases} 2\sum_{j=0}^{n-1}\cos(k'j\pi/n)b_j + (-1)^{k'}b_n & \text{for} \quad k = 2k' \\ 2i\sum_{j=1}^{n-1}\sin((k'-1/2)j\pi/n)b_j + (-1)^{k'-1}ib_n & \text{for} \quad k = 2k'-1 \end{cases}. \end{aligned}$$

Thus if b_0 is doubled, the even and odd k values are given by cosine and shifted sine transforms respectively:

$$\begin{aligned} \hat{x}(2k) &= 2\left(T_{NN}\mathbf{b}\right)(k) \\ \hat{x}(2k-1) &= 2i\left(T_{DN}\mathbf{b}\right)(k). \end{aligned} \qquad \text{(D.99)}$$

Real arithmetical operation counts, as coded, for the transform algorithms described above are given in table D.3.

Length N Transforms	Real Operation Count
Complex FFT	$(5 \log_2 N)N$
Paired Real	$(5 + 5 \log_2 N)N$
Paired Sine	$(13 + 5 \log_2 N)N$
Paired Cosine	$(14 + 5 \log_2 N)N$
Single real with zero-extended data†	$(14 + 5 \log_2 N)N$
Paired real with symmetric, zero-extended data‡	$(30 + 10 \log_2 N)N$
Paired shifted sine or cosine	$(12 + 5 \log_2 N)N$
Paired dual shifted sine or cosine	$(18 + 5 \log_2 N)N$

†Data is $b_0, ..., b_N, 0, ..., 0$ (length $2N$).
‡Data is $b_0, ..., b_{N-1}, b_N, b_{N-1}, ..., b_0, 0, ..., 0$ (length $4n$).

Table D.3: Transform Operation Counts

D.3.3 Implementation of the James Algorithm

The basic theory of the algorithm described in section D.1.2 is straightforward, but several complications arise during the implementation. These come from the handling of the 6 planes that constitute the boundary of the computational box R, especially in phase 2 (boundary convolution). In addition, the presence of one or more planes of symmetry entails major modifications.

The present implementation differs considerably from the description of James[89], whose method uses complex manipulations of sine and cosine transforms to conserve memory during the boundary convolution. We the process by combining these transforms into standard real and complex transforms. Several extra scratch planes are used to achieve major gains in vectorization, as well as simplification.

At the highest level the algorithm is divided into three separate, but parallel, paths for (1) no planes of symmetry (2) only a Y plane of symmetry (3) both Y and Z planes of symmetry. If there is only a Z plane of symmetry, the Y and Z data are interchanged before and after doing the Y only algorithm. Within paths (2) and (3) there is a choice as to whether a plane of symmetry is located on the left (origin) or right. For example, if the computational box is represented by

$$R = [0, m_x] \times [0, m_y] \times [0, m_z]$$

with coordinates i, j, k, then a Y plane of symmetry may be specified either by $j = 0$ or $j = m_y$. The algorithm assumes the $j = m_y$ case, and simply reverse the Y data before and after if it is the $j = 0$ case. The algorithm could be formulated the other way around just as easily.

Phase 1

The basic method used to solve the Dirichlet problem (D.71) for the interior solution θ is to use X and Z sine transforms and a tridiagonal solver, as in formula (D.88). However, instead of completing the interior solution in phase 1, the inverse X and Z sine transforms are delayed to phase 3, where they are combined with the inverse X and Z transforms for ψ. This is possible because phase 2 requires only the boundary charge function σ, which in turn needs only θ values next to the edge of the box. For example, the computation of σ on the $i = 0$ plane, denoted by σ_{yz0}, reduces to

$$\begin{aligned} \sigma_{yz0}(j,k) &= Q(j,k) - (D^2\theta)(0,j,k) \\ &= Q(j,k) - \theta(1,j,k)/\Delta x^2 \end{aligned} \tag{D.100}$$

since θ is zero outside R and on ∂R.

Using α, β, γ to indicate sine transformed coordinates, $\theta(1, j, k)$ can be computed explicitly by

$$\theta(1,j,\gamma) = \frac{2}{m_x} \sum_{\alpha=1}^{m_x-1} \sin\left(\alpha \frac{\pi}{m_x}\right) \theta(\alpha,j,\gamma). \tag{D.101}$$

Then an inverse sine transform in Z is applied to get $\theta(1, j, k)$. Similarly

$$\theta(m_x-1,j,\gamma) = \frac{2}{m_x} \sum_{\alpha=1}^{m_x-1} (-1)^{\alpha-1} \sin\left(\alpha \frac{\pi}{m_x}\right) \theta(\alpha,j,\gamma). \tag{D.102}$$

Actually it is better to replace (D.101) and (D.102) by

$$\begin{aligned} e(j,\gamma) &= \frac{2}{m_x}\sum_{\alpha=1}^{n_e}\sin\left(2\alpha\frac{\pi}{m_x}\right)\theta(2\alpha,j,\gamma) \\ o(j,\gamma) &= \frac{2}{m_x}\sum_{\alpha=1}^{n_o}\sin\left((2\alpha-1)\frac{\pi}{m_x}\right)\theta(2\alpha-1,j,\gamma) \end{aligned} \tag{D.103}$$

$$\begin{aligned} \theta(1,j,\gamma) &= o(j,\gamma)+e(j,\gamma) \\ \theta(m_x-1,j,\gamma) &= o(j,\gamma)-e(j,\gamma) \end{aligned}$$

for $n_e = (m_x-1)/2$ and $n_o = m_x/2$. Of course the sine factors in (D.40) are precomputed.

If there is a Z plane of symmetry, the major difference is that Z sine transforms are replace by shifted sine transforms, according to table D.2, since the symmetry plane (= zero Neumann boundary condition) is assumed to be on the right. This also means that the sine term in (D.101) is replaced by the shifted sine term $\sin((\alpha-\frac{1}{2})\pi/m_z)$ when computing $\theta(\alpha,j,1)$.

If there is a Y plane of symmetry, the major difference is in the tridiagonal solver. By (D.73) there is a 2 in the subdiagonal of the bottom row, so the algorithm described in section D.3.1 is modified accordingly.

Phase 2

The basic idea of the boundary convolution is to zero extend the box R, doubling its size in each dimension, in order to compute ψ on ∂R by

$$\psi = G*\sigma + G_d*\tau = (\hat{G}\hat{\sigma}+\hat{G}_d\hat{\tau})^\vee. \tag{D.104}$$

According to (D.41) the downstream source convolution $G_d*\tau$ is to be interpreted in the following way. Let F_x = forward DFT in X, etc. Then

$$\begin{aligned} (G_d*\tau)(i,j,k) &\equiv (G_d(i,\cdot,\cdot))*\tau(j,k) \\ &= F_z^{-1}F_y^{-1}\left((F_x^{-1}\hat{G}_d)(F_yF_z\tau)\right) \\ &= F_z^{-1}F_y^{-1}F_x^{-1}\left(\hat{G}_d(F_yF_z\tau)\right). \end{aligned} \tag{D.105}$$

This means that FFT's are first applied to the 6 boundary planes of σ. Then, in a loop over the Z index, an XY plane of $\hat{\sigma}$ is assembled from the 6 transformed planes. At the same time an XY plane of $\hat{G}$ is assembled from its 3 plane representation (D.37) and a plane of $\hat{G}_d$ from its 4 plane representation (D.49). Then the multiplications and additions in (D.104) are done to get $\hat{\psi}$. Still inside the Z loop, some of the inverse DFT's are performed explicitly to get 6 boundary planes. After the Z loop, the remaining inverse DFT's are applied to these boundary planes to get $\psi|_{\partial R}$.

Let's look at the transformation of the YZ boundary planes, σ_{yz0} and σ_{yz1}, in more detail. Since these are real, we may start with a real transform in Z. Next apply a complex transform in Y. The transform in X reduces to a butterfly operation (addition

and subtraction), with the sum of the two planes giving the even frequencies and the difference the odd frequencies:

$$\hat{\sigma}_{yz}(\beta,\gamma,p_\alpha) \equiv \hat{\sigma}_{yz0}(\beta,\gamma) + (-1)^\alpha \hat{\sigma}_{yz1}(\beta,\gamma) \tag{D.106}$$

for $p_\alpha = \alpha \bmod 2$.

If there is a Y plane of symmetry, the Y transform becomes a transform of symmetric, zero-extended data with frequencies from 0 to $2m_y$. By (D.99) the odd frequencies may be represented as nominally real values, like the even frequencies, but with an implicit scaling by i. If this is applied to the single XZ boundary plane, the result is just scaling by 2 for even frequencies and zero for odd frequencies:

$$\hat{\sigma}_{xz}(\alpha,\gamma) := \begin{cases} 2\hat{\sigma}_{xz}(\alpha,\gamma) & \text{for } \beta \text{ even} \\ 0 & \text{for } \beta \text{ odd.} \end{cases} \tag{D.107}$$

The same methods apply to the transforms and inverse transforms of the other boundary planes.

By (D.106) an XY plane of $\hat{\sigma}$ may be assembled by the following formula:

$$\hat{\sigma}(\alpha,\beta,\gamma) = \hat{\sigma}_{yz}(\beta,\gamma,p_\alpha) + \hat{\sigma}_{xz}(\alpha,\gamma,p_\beta) + \hat{\sigma}_{xy}(\alpha,\beta,p_\gamma). \tag{D.108}$$

One way to do this is by adding the XY and XZ planes while vectorizing in X:

$$t(\alpha,\beta) = \hat{\sigma}_{xz}(\alpha,\gamma,p_\beta) + \hat{\sigma}_{xy}(\alpha,\beta,p_\gamma). \tag{D.109}$$

Then complete (D.107) and at the same time use (D.104) and (D.105) to do the Green's function multiplications, taking advantage of operation chaining while vectorizing in Y:

$$\hat{\psi}(\alpha,\beta,\gamma) = (\hat{\sigma}_{yz}(\beta,\gamma,p_\alpha) + t(\alpha,\beta))\,\hat{G}(\alpha,\beta,\gamma) + \hat{\tau}(\beta,\gamma)\hat{G}_d(\alpha,\beta,\gamma) \tag{D.110}$$

If there is a Y plane of symmetry, (D.107) shows that (D.109) is just a copy for odd β and addition of the single XZ plane for even β. If there is also a Z plane of symmetry, (D.109) is only addition for β and γ both even, and is zero if both are odd. Furthermore, in (D.110) $\hat{\sigma}_{yz}$ and $\hat{\tau}$ are nominally real, with only implicit scaling by i for odd β and γ. Thus the nominally imaginary part of (D.110) reduces to

$$\mathrm{Im}(\hat{\psi}(\alpha,\beta,\gamma)) := \mathrm{Im}(t(\alpha,\beta))\hat{G}(\alpha,\beta,\gamma). \tag{D.111}$$

According to the Z inverse DFT formula, the boundary XY planes may be computed from $\hat{\psi}$ by butterflying the sums of the even and odd Z index values:

$$\begin{aligned} e(\alpha,\beta) &= \frac{1}{2m_z}\sum_{\gamma=0}^{m_z-1} \hat{\psi}(\alpha,\beta,2\gamma) \\ o(\alpha,\beta) &= \frac{1}{2m_z}\sum_{\gamma=0}^{m_z-1} \hat{\psi}(\alpha,\beta,2\gamma+1) \end{aligned} \tag{D.112}$$

$$\begin{aligned} \hat{\psi}_{xy0}(\alpha,\beta) &= e(\alpha,\beta) + o(\alpha,\beta) \\ \hat{\psi}_{xy1}(\alpha,\beta) &= e(\alpha,\beta) - o(\alpha,\beta) \end{aligned} \tag{D.113}$$

The sums in (D.112) are accumulated in the Z loop with the butterflies and scaling done later as an inverse operation to the butterflies (D.106).

A way to gain efficiency is to let the Z loop run only from 1 to $m_z - 1$ and set

$$\hat{\psi}(\alpha, \beta, 2m_z - \gamma) = \bar{\hat{\psi}}(2m_x - \alpha, 2m_y - \beta, \gamma), \tag{D.114}$$

which follows directly from the DFT formula. In the case of Y symmetry (D.114) may be replaced by

$$\hat{\psi}(\alpha, \beta, 2m_z - \gamma) = (-1)^{\beta} \bar{\hat{\psi}}(2m_x - \alpha, \beta, \gamma) \tag{D.115}$$

according to formula (D.99). In case of Z symmetry, the loop runs from 0 through $2m_z$ and (D.114) may be replaced by

$$\hat{\psi}(\alpha, \beta, 2m_z - \gamma) = (-1)^{\gamma} \hat{\psi}(\alpha, \beta, \gamma). \tag{D.116}$$

That is, the odd frequencies sum to zero and the even ones are doubled except at the endpoints 0 and $2m_z$.

The boundary XZ and XY planes are computed by the same principle of summing even and odd index values, but with complete computation of an X or Y line done all at once each time through the Z loop. For example, if there is a Y plane of symmetry, the even and odd β sums of $\hat{\psi}$ reduce to just doubling the even values, except for endpoints, by the Y symmetry analog of (D.116).

Phase 3

The method used in phase 1 is applied again in phase 3, but specialized to zero interior sources. This means that, for example, the two YZ boundary planes ψ_{yz0} and ψ_{yz1} are converted to equivalent sources at positions $i = 1$ and $i = m_x - 1$ respectively by scaling by $-1/\Delta x^2$, according to (D.71). Now sine transform in Z to get γ coordinates. Next, the sine transform in X to get α coordinates reduces to the following.

$$\psi_{yz}(\alpha, j, \gamma, p_\alpha) = \frac{-1}{\Delta x^2} \sin\left(\alpha \frac{\pi}{m_x}\right) (\psi_{yz0}(j, \gamma) + (-1)^{\alpha} \psi_{yz1}(j, \gamma)) \tag{D.117}$$

for $p_\alpha = \alpha \bmod 2$. The butterfly operations in (D.117) are done before the Z loop, and the precomputed sine factors are multiplied inside the loop. There is a formula analogous to (D.117) for the transformed XY boundary planes $\psi_{xy}(\alpha, j, \gamma, p_\gamma)$. The XZ boundary planes require full X and Z sine transforms plus scaling by $-1/\Delta y^2$. Denote them by $\psi_{xz}(\alpha, \gamma, 0)$ and $\psi_{xz}(\alpha, \gamma, 1)$. Then inside the Z loop the transforms of the 6 boundary planes are summed to get an XY plane t as follows.

$$\begin{aligned} t(\alpha, j) \;=\; & \psi_{yz}(\alpha, j, \gamma, p_\alpha) + \psi_{xy}(\alpha, j, \gamma, p_\gamma) + \\ & \delta(j-1)\psi_{xz}(\alpha, \gamma, 0) + \delta(j - m_y - 1)\psi_{xz}(\alpha, \gamma, 1). \end{aligned} \tag{D.118}$$

In the case of Y symmetry, there is only one XZ plane. In the case of Z symmetry there is only one XY plane, and shifted sine transforms are used. The tridiagonal solver is applied to the plane t, and the result is added to the corresponding plane of $\theta(\alpha, j, \gamma)$, saved from phase 1.

After the Z loop, the inverse X and Z transforms are applied to get ϕ in the interior of the box, and the original 6 boundary planes are copied onto its boundary.

Appendix E

SPARSE SOLVER

In this appendix, the general purpose sparse solver that is used to factor the sparse matrix is discussed. The sparse matrix is used as a left preconditioner in the solution of the discrete equations, see Section 2.4. The sparse solver was designed for general usage to solve much larger problems than are feasible with existing sparse matrix software. The solver has a general input capability allowing contributions to matrix elements to be entered in any order. These contributions are sorted and combined to produce the final matrix. This feature is particularly convenient with finite elements, where element stiffness matrices can be generated in any order. The solver is out-of-core so that quite large problems can be solved on current computers. Gaussian elimination is performed by block rows, additional blocks being created as fill is generated. The sparse solver takes full advantage of the hardware features on Cray computers including gather/scatter, vector compress, and large out-of-core memory afforded by the SSD on the Cray X-MP or Cray Y-MP. Considerable attention has been devoted to making all phases of setting up and solving matrix problems convenient and efficient. A description of these phases including ordering the matrix elements to minimize the fill-in, matrix assembly, matrix decomposition, and the forward and backward substitution is presented.

E.1 NESTED DISSECTION ORDERING

For large sparse problems, a matrix decomposition preconditioner is practical only if the decomposition is also sparse. This is true not only because of storage limitations, but also because of the CPU time required for forward and back substitution. One key to maintaining sparsity is a good permutation ordering for the rows and columns of the matrix. For sparse matrices resulting from standard discretizations of elliptic partial differential equations on uniform rectangular grids nested dissection has been shown to be asymptotically optimal [97].

In TRANAIR a physically based version of nested dissection suitable for grids with local refinements has been implemented. One advantage of this method is that it does not require an examination of the graph of the matrix. The algorithm acts recursively on subsets of nodes (grid points). In TRANAIR, the discretization used

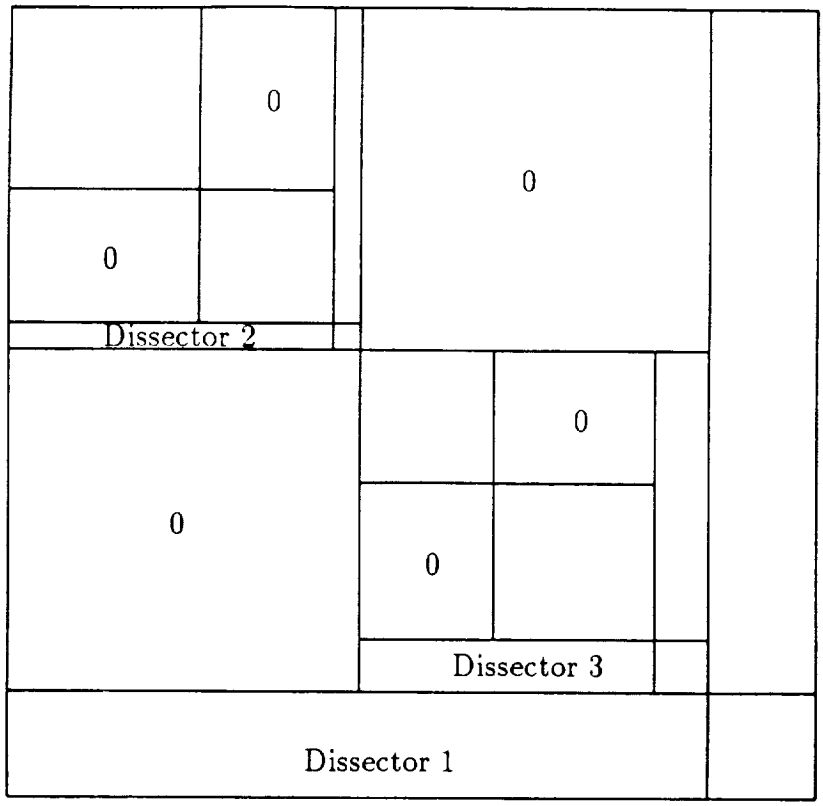


Figure E.1: Block Structure of a Sparse Matrix Ordered with Nested Dissection.

near boundary surfaces can locate several solution unknowns at a same grid node location. The first such subset is the set of all nodes in the reduced set. For a set of nodes $\mathcal{N}$ the algorithm finds a set of nodes $\mathcal{N}_d$ called a dissector. Writing $\mathcal{N} = \mathcal{N}_d \cup \mathcal{N}_l \cup \mathcal{N}_r$, where $\mathcal{N}_l$ consists of nodes on one side of $\mathcal{N}_d$ and $\mathcal{N}_r$ those on the other. The dissector has the property that an unknown at any node on one side has a stencil that does not include unknowns located at nodes on the other side. The permutation is produced by ordering the nodes in the dissector last. Figure E.1 shows the block structure of this matrix. The blocks of zeros remain intact, preserving sparsity during the decomposition. For a structured grid a plane of points forms a suitable dissector for a standard 27 point stencil.

In TRANAIR, dissectors are generated by first taking a cutting plane perpendicular to a coordinate axis and finding the set $\mathcal{B}$ of all boxes intersecting this plane by interrogating the oct-tree data structure (see Appendix A). Taking the case when the cutting plane is perpendicular to the x axis, the dissector of all nodes on the left hand (negative x) face of boxes in $\mathcal{B}$ that are also in $\mathcal{N}$. This will provide a dissector except near pseudo-nodes where the stencil is altered (see Section 2.3.5). When a pseudo-node is in the dissector its parent nodes must also be included in the dissector. Figure E.2 shows examples (in two dimensions) of cutting planes and

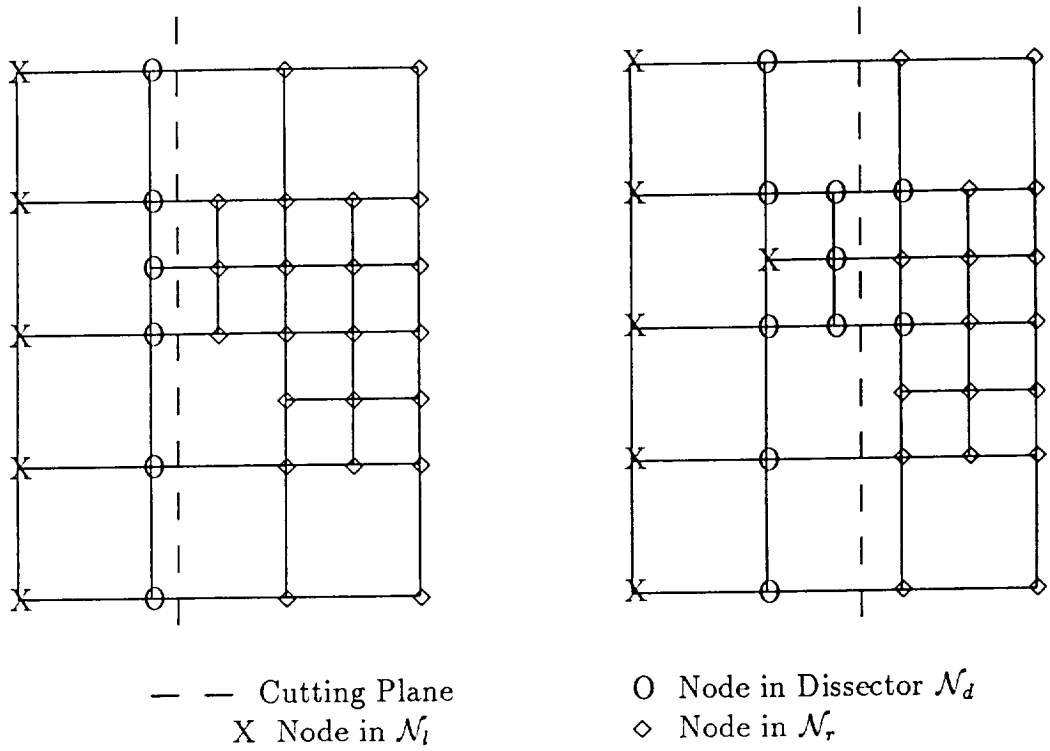


Figure E.2: Examples of Cutting Planes and Nodes in Resulting Dissector.

corresponding sets of nodes in the dissectors.

There are two guiding principles that aid in producing an effective nested dissection ordering. The first is that the components $\mathcal{N}_l$ and $\mathcal{N}_r$ resulting from the dissection be of approximately equal size. The second is that the dissectors contain as few unknowns as possible. (The size of the dissectors can vary due to local refinement and the location of boundaries.) These principles can conflict and some compromise is necessary. The cutting plane is selected to have x coordinate equal to that of the node in $\mathcal{N}$ with median x coordinate. Dissectors perpendicular to each of the three coordinate axes are tested and the one yielding the smallest dissector is chosen. The process is repeated recursively on the newly formed components resulting from each dissection until all remaining components contain fewer than 50 nodes.

The above algorithm is modified when regions of supersonic flow are present in the full potential case because upwinding of the density enlarges the stencil (see Section 2.3.7). If the cutting plane intersects a box which contains supersonic flow or is adjacent to such a box then all the nodes of the box are included in the dissector.

E.2 MATRIX ASSEMBLY

Contributions to the global stiffness matrix are generated on an element by element basis, i.e., element stiffness matrices are input one by one. This order is unrelated to

the ordering of the unknowns used for the decomposition. Thus, the contributions must be sorted and coalesced to produce the final global stiffness matrix. This process consists of four steps and is illustrated in the case of two blocks and a 3 by 3 matrix in Figure E.3.

Each contribution is described by a numerical value (denoted by a letter) and by row and column indices. The four steps are as follows:

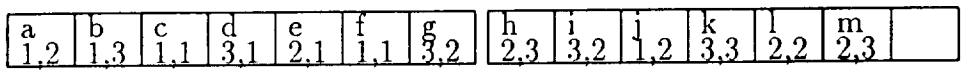
- 1. All contributions are collected in equal sized blocks and stored out of core.
- 2. Each block is returned to main memory and sorted by row index, resulting in an ordered sequence of row groups within each block. A row group is a group of contributions all having the same row index. A simple bucket sort algorithm [98] seems to be the most efficient for this purpose.
- 3. These sorted blocks are then merged into ordered chains of blocks. An ordered chain is a chain of blocks such that all contributions in a given block have row indices less than or equal to those in all subsequent contributions in that block and the remaining blocks of the chain. Merging two chains consists in interleaving their row groups so that the resulting chain is also sorted by row index. At a given stage, the two shortest chains are always merged. Ultimately, the result is a single chain of blocks. Because the contributions are not sorted by column index at this stage, all movement of elements can be done by row group. This allows vectorization of the merge algorithm.
- 4. All contributions to the same matrix element are then coalesced, i.e., within each row group, contributions with common column indices are added to form a single contribution. Coalescing can be done without first sorting each row group by column index.

Note that most elements of the global stiffness matrix have contributions from 8 element stiffness matrices in subsonic regions and as many as 64 in supersonic regions. Thus, the number of contributions may be up to 125 times greater than the number of elements in the assembled global stiffness matrix. In order to minimize storage requirements, the above four step process is performed repeatedly. (It can be performed at any point in the generation of contributions to the stiffness matrix.) When this is done, chains which have already been formed are not merged until new chains of equal size exist. Row groups are sorted by column index using a bucket sort only after completion of contribution input and coalescing.

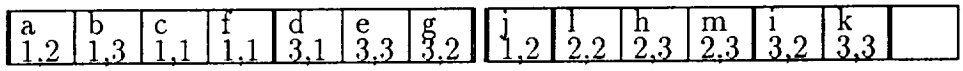
E.3 MATRIX DECOMPOSITION

During the decomposition phase, the matrix is stored in a row format. For the purposes of transferring information between main memory and the SSD, the matrix is partitioned into row blocks. For each element of the matrix, two storage locations are used, one to store the element and the other to store its column index. Optionally, the matrix element and the column index can be packed into one word of storage. In

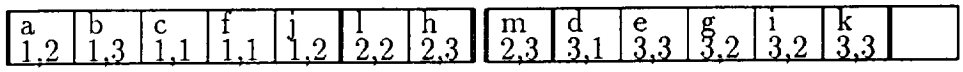
Contributions input in two blocks



Sort each block into row groups



Merge two blocks into a chain of blocks



Coalesce all contributions to the same matrix element

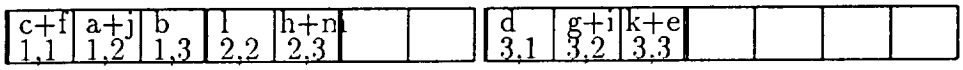


Figure E.3: Sorting and Merging Procedure.

this mode, the 64-bit storage location devotes 43 bits to the matrix element and 21 bits to its column index. Word packing reduces the SSD storage required to hold the matrix decomposition. Moreover, the CPU time required to perform the packing and unpacking is compensated for by the reduced time spent referencing memory. As a result, the word packed version of the sparse solver actually runs slightly faster than the unpacked version.

Decomposition of the matrix is accomplished by Gaussian elimination. (For sparse matrix problems arising in TRANAIR no pivoting has yet been found necessary for numerical stability.) Each element in the lower triangle is eliminated in turn through a sparse SAXPY (SPAXPY) operation with the appropriate row. The multiplier becomes the corresponding element of the L matrix, and the appropriate U matrix row is modified by the SPAXPY operation. Each row block is decomposed, and when finished, used to eliminate corresponding lower triangular elements from all subsequent row blocks. As fill-in occurs the row blocks must be repartitioned and new row blocks added. An input/output package has been developed that does this automatically so that formally the code need only fetch or store any given row. When all lower triangular elements of a given row have been eliminated, small elements in the upper triangular part of the row can be dropped if they are small relative to the current row or column diagonal for that element. The criterion chosen depends on whether the problem is deemed well scaled row-wise or column-wise. The lower triangular elements are dropped in a similar fashion as they are eliminated, obviating the need for a SPAXPY operation.

Unlike many sparse matrix solvers, there is no reliance on a symbolic factorization to facilitate the matrix decomposition. Instead, an explicit search is carried out to find the nonzero elements created during the matrix decomposition. This strategy allows the easy implementation of drop tolerances as required for the large problems

discussed in the following sections.

As pointed out in the sparse matrix literature (for example, [99]), searching for nonzero elements would be prohibitively expensive on a normal scalar machine. Fortunately, the hardware vector mask and vector compress feature on the Cray X-MP allows us to perform the searches efficiently. For the largest problems solved to date involving extensive use of drop tolerances (those problems incurring the maximum penalty for nonzero searching), about 40 percent of the total decomposition cost is taken by the searches for the next nonzero element below the diagonal to be eliminated. Another 40 percent of the cost of the decomposition is taken by the SPAXPY operations which would be required whether or not a symbolic factorization was available. The final 20 percent is taken by searching for the nonzero elements in the upper part of the matrix created by the SPAXPY operations. (The number of searches of this type depends on the size of blocks that can be held in core relative to the size of the LU decomposition. This 20 percent can be thought of as penalty for being out-of-core.) Thus in the worse case, for the moderately sparse matrices encountered in applications, the searches for nonzero elements increase the cost by about a factor of two. This cost is more than offset by the ability to introduce a drop tolerance.

E.4 FORWARD/BACKWARD SUBSTITUTION

The final phase is the forward/backward substitution. For problems requiring solutions for many right-hand sides, the substitution phase can be more expensive than the decomposition phase. Therefore, it is important to minimize the cost of the forward/backward substitution phase. For the short vector lengths characteristic of sparse matrix operations, a sparse vector dot product typically takes twice as long as a SPAXPY on the Cray X-MP. To take advantage of the relative speed of SPAXPY operations during the solution phase, instead of solving $Ax = b$, the equivalent system $x^t A^t = b^t$ is solved. Specification of the transposed problem is easily accomplished by transposing the row and column indices as they are collected.

E.5 PERFORMANCE

Typically, the matrices have between 30,000 and 300,000 rows with 20–30 nonzeros per row in subsonic flow. In regions of supersonic flow, there are 100–120 nonzeros per row. This increase is due to the larger operator stencil necessary to include the upwinding needed to rule out expansion shocks [22]. Without the use of a drop tolerance, significant fill-in occurs during the decomposition yielding a decomposed matrix with 10–30 times the number of nonzero entries in the original matrix. For many large problems, the memory required by a full decomposition would be too large for the SSD. Because the matrix is used as a preconditioner to solve the problem iteratively, it is not essential that the decomposition be exact. During the decomposition elements are dropped when they are less than a specified fraction (the drop tolerance) of the current diagonal element . With a suitable choice of drop tolerance, CPU time

and memory use for the decomposition are reduced by up to an order of magnitude with only a slight degradation in convergence rate.

Table E.1: Performance Characteristics for the Sparse Solver with No Drop Tolerance. Ten to Twenty Nonlinear Newton Steps are Required for each Solution. Each Linearized Solution Requires about 10 GMRES Iterations.

Matrix equations	Decomp CPU sec	CPU sec per iteration	Decomp (MW)
10,018	20	1	3
30,267	280	3	11
36,627	230	4	16
50,655	620	3	29
63,069	420	6	26

Table E.2: Performance Characteristics for the Sparse Solver with Drop Tolerance. Each Linearized Solution Requires About 20–40 GMRES Iterations.

Total equations	Matrix equations	Drop tolerance	Decomp CPU sec	CPU second per iteration	Decomp (MW)
18,376	11,514	0.0010	7	0.2	1.5
24,304	15,051	0.0010	11	0.4	2.2
37,702	18,731	0.0010	13	0.4	2.4
61,863	44,537	0.0010	41	0.9	6.7
124,878	81,216	0.0005	91	1.3	12.6
236,970	167,500	0.0010	246	2.8	26.4
247,703	156,659	0.0008	241	3.3	28.6
268,301	192,238	0.0010	311	3.1	30.8
284,052	181,802	0.0008	297	4.1	30.6
288,333	207,827	0.0010	405	3.5	40.0
373,813	241,474	0.0010	631	5.8	54.0
480,907	330,857	0.0008	882	6.8	64.6

Tables E.1 and E.2 give the computer times for decomposition of the reduced set matrix for several representative cases run TRANAIR with and without a drop tolerance as well as the computer time required for each GMRES iteration. The storage required for the decomposition (using no word packing) is also shown and is equal to twice the number of nonzero entries in the decomposition.

Table E.2 gives both the number of equations in the reduced set (the size of the problem given to the sparse solver) and the total number of degrees of freedom. The size of the sparse matrix varies from about 10,000 unknowns to around 330,000 unknowns. Note that with the use of a drop tolerance, the decomposition costs are

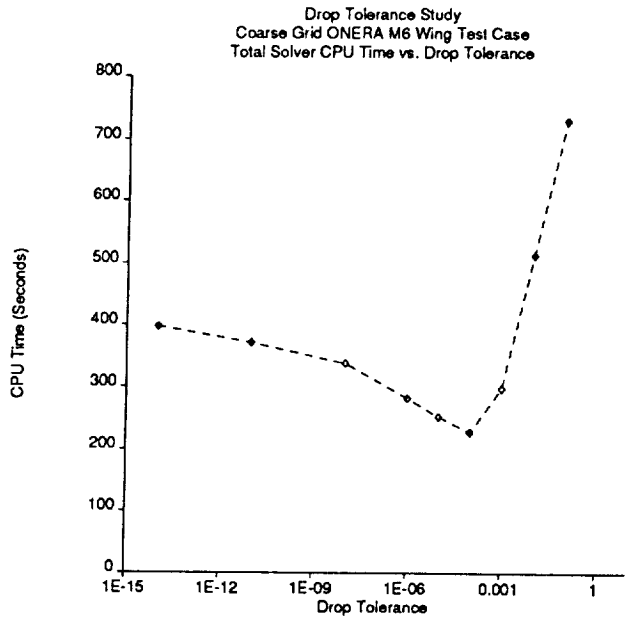


Figure E.4: Cost versus drop tolerance for the ONERA M6 TRANAIR solution.

reduced by nearly an order of magnitude. The costs per iteration are also reduced by about a factor of two. For each linear problem 10–20 iterations required to converge the solution when no drop tolerance is used. By using a drop tolerance in the range 0.001–0.0001 these numbers are increased to 20–40 iterations. However, since the cost per iteration has been reduced by a factor of two, this helps to compensate for the increase in number of iterations. For drop tolerance in this range the size of the decomposition is between two to five times the size of the original matrix.

Memory limitations make it impossible to test the effect of a full range of drop tolerances for a large problem. However, for a small problem with 28,050 finite elements serves to illustrate the effect of drop tolerance on the decomposition and overall solution costs. Figure E.4 illustrates total solution costs as a function of drop tolerance for a fluid dynamics problem (an ONERA M6 wing in transonic flow).

Figure E.5 illustrates the SSD resource requirements. It is clear that there is a minimum in the total cost for drop tolerances in the range of 10^{-4}. When higher drop tolerances are used, the decomposition costs continue to decrease, but the iterative costs begin to increase due to the larger number of iterations required to reach a given level of convergence. For this configuration, a drop tolerance in the range of 10^{-3} produces a slight increase in CPU time over the optimal value, but (Fig. E.5) significantly reduces the amount of SSD storage required (by more than a factor of two). CPU time must be balanced against the SSD storage requirements. Thus some experimentation may be required to determine an appropriate value of drop tolerance. For larger problems, the amount of SSD storage may become the critical limiting factor. The reduction in SSD storage for larger problems can be as high as a factor of 20 with an appropriate choice of drop tolerance.

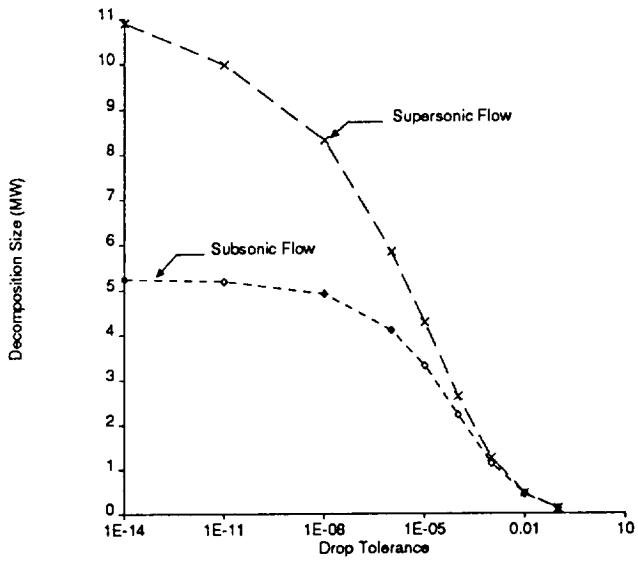


Figure E.5: SSD storage versus drop tolerance for the ONERA M6 TRANAIR solution.

References

[1] Goldhammer, M. I.; and Rubbert, P. E.: CFD in Design—An Airframe Perspective. AIAA Paper 89-0092, Jan. 1989.

[2] Tinoco, E. N.; and Rubbert, P. E.: Impact of Computational Aerodynamics on Aircraft Design. AIAA Paper 83-2060, Aug. 1983.

[3] Miranda, L. R.: Transonics and Fighter Aircraft: Challenges and Opportunities for CFD. NASA CP-3020, vol. 1, part 1, 1989, pp. 153-173.

[4] Hess, J. L.; and Smith, A. M. O.: Calculation of Nonlifting Potential Flow about Arbitrary Three Dimensional Bodies. ES40622, Douglas Aircraft Co., Long Beach, Calif., 1962.

[5] Rubbert, P. E.; and Saaris, G. R.: Review and Evaluation of a Three Dimensional Lifting Potential Flow Computational Method for Arbitrary Configurations. AIAA Paper 72-188, Jan. 1972.

[6] Roberts, A.; and Rundle, K.: Computation of First Order Compressible Flow about Wing-Body Configurations. (Available to U.S. Government agencies only.) S/T-MEMO-14173, British Aircraft Co., 1973.

[7] Morino, L.; and Kuo, C.-C.: Subsonic Potential Aerodynamics for Complex Configurations: General Theory. AIAA Journal, vol. 12, no. 2, 1974, pp. 191-197.

[8] Johnson, F. T.; and Rubbert, P. E.: Advanced Panel-Type Influence Coefficient Methods Applied to Subsonic Flows. AIAA Paper 75-50, Jan. 1975.

[9] Bristow, D. R.; and Grose, G. G.: Modification of the Douglas Neumann Program to Improve the Efficiency of Predicting Component Interference and High Lift Characteristics. NASA CR-3020, 1978.

[10] Johnson, F. T.: A General Panel Method for the Analysis and Design of Arbitrary Configurations in Incompressible Flows: Boundary Layer Problem. NASA CR-3079, 1980.

[11] Ehlers, F. E.; Epton, M. A.; Johnson, R. T.; Magnus, A. E.; and Rubbrt, P. E.: A Higher Order Panel Method for Linearized Supersonic Flow. NASA CR-3062, 1979.

[12] Carmichael, R. L.; and Erickson, L. L.: PANAIR: A Higher Order Panel Method for Predicting Subsonic or Supersonic Linear Potential Flows about Arbitrary Configurations. AIAA Paper 81-1255, June 1981.

[13] Dusto, A. R.; and Epton, M. A.: An Advanced Panel Method for Analysis of Arbitrary Configurations in Unsteady Subsonic Flow. NASA CR-152323, 1980.

[14] Rowe, W. S.; Winther, B. A.; and Redman, M. C.: Prediction of Unsteady Aerodynamic Loadings Caused by Trailing Edge Control Surface Motions in Subsonic Compressible Flow–Analysis and Results. NASA CR-2003, 1972.

[15] Tinoco, E. N.; Ball, C. N.; and Rice, F. A. II: PAN AIR Analysis of a Transport High Lift Configuration. AIAA Paper 86-1811, June 1986.

[16] Dusto, A. R.: Aerodynamic Analysis of a Fighter Aircraft with a Higher Order Paneling Method. Technical Report AFWAL-TR-80-3115, Wright-Patterson Air Force Base, Ohio, 1980.

[17] Murman, E. M.; and Cole, J. D.: Calculation of Plane Steady Transonic Flows. AIAA Journal, vol. 9, no. 1, 1971, pp. 114-121.

[18] Bailey, F. R.; and Ballhaus, W. F.: Relaxation Methods for Transonic Flow about Wing-Cylinder Combinations and Lifting Swept Wings. Proc. Third International Congress on Numerical Methods in Fluid Dynamics, Volume 2, Springer-Verlag, 1972, pp. 2-9.

[19] Jameson, A.: Iterative Solution of Transonic Flows Over Airfoils and Wings, Including Flows at Mach 1. Communications on Pure and Applied Mathematics, vol. 27, no. 3, 1974, pp. 283-309.

[20] Caughey, D. A.; and Jameson, A.: Numerical Calculation of Transonic Potential Flow About Wing-Fuselage Combinations. AIAA Paper 77-677, June 1977.

[21] Ballhaus, W. F.; Jameson, A.; and Albert, J.: Implicit Approximate-Factorization Schemes for the Efficient Solution of Steady, Transonic Flow Problems. AIAA Paper 77-634, June 1977, pp. 27-34.

[22] Hafez, M. M.; Murman, E. M.; and South, J. C.: Artificial Compressibility Methods for Numerical Solutions of Transonic Full Potential Equation. AIAA Paper 78-1148, July 1978.

[23] Bristeau, M. O.; Glowinski, R.; Periaux, J.; Perrier, P.; Pironneau, O.; and Poirier, G.: Application of Optimal Control and Finite Element Methods to the Calculation of Transonic Flows and Incompressible Viscous Flows. Numerical Methods in Applied Fluid Dynamics, B. Hunt, ed., Academic Press, London, 1980, pp. 203-312.

[24] Holst, T. L.: A Fast, Conservative Algorithm for Solving the Transonic Full-Potential Equation. AIAA Paper 79-1456, July 1979.

[25] Holst, T. L.; and Ballhaus, W. F.: Fast, Conservative Schemes for the Full Potential Equation Applied to Transonic Flows. AIAA Journal, vol. 17, 1979, pp. 145-152.

[26] Boppe, C. W.; and Stern, M. A.: Simulated Transonic Flows for Aircraft with Nacelles, Pylons and Winglets. AIAA Paper 80-0130, Jan. 1980.

[27] Yu, N. J.: Grid Generation and Transonic Flow Calculations for Three Dimensional Configurations. AIAA Paper 80-1391, July 1980.

[28] Lee, K. D.: 3-D Transonic Flow Computations Using Grid Systems with Block Structure. AIAA Paper 81-0998, June 1981.

[29] Jameson, A., Schmidt, W., and Turkel, E.: Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes. AIAA Paper 81-1259, June 1981.

[30] Chakravarthy, S. R.; and Osher, S.: A New Class of High Accuracy TVD Schemes for Hyperbolic Conservation Laws. AIAA Paper 85-0363, Jan. 1985.

[31] Pulliam, T. H.: Euler and Thin Layer Navier-Stokes Codes: ARC2D, ARC3D. Computational Fluid Dynamics, K. C. Reddy and J. S. Steinhoff, eds., University of Tennessee Space Institute, Publication No. E02-4005-023-84, Tullahoma, TN, 1984, pp. 15.1-15.85.

[32] MacCormack, R. W.: Current Status of Numerical Solutions of the Navier-Stokes Equations. AIAA Paper 85-0032, Jan. 1985.

[33] Thomas, J. L.; and Walters, R. W.: Upwind Relaxation Algorithms for the Navier-Stokes Equations. AIAA Paper 85-1501, July 1985.

[34] Jameson, A.: Transonic Flow Calculations. Department of Mechanical and Aerospace Engineering, Report No. 1651, Princeton University, 1983.

[35] Martinelli, L.; Jameson, A.; and Grasso, F.: A Multigrid Method for the Navier-Stokes Equations. AIAA Paper 86-0208, Jan. 1986.

[36] Rubbert, P. E.; Bussoletti, J. E.; Johnson, F. T.; Sidewell, K. W.; Rowe, W. S.; Samant, S. S.; SenGupta, G.; Weatherill, W. H.; Burkhart, R. H.; Everson, B. L.; Young, D. P.; and Woo, A. C.: A New Approach to the Solution of Boundary Value Problems Involving Complex Configurations. Computational Mechanics–Advances and Trends, Ahmed K. Noor, ed., the American Society of Mechanical Engineers, New York, 1986, pp. 49-84.

[37] Samant, S. S.; Bussoletti, J. E.; Johnson, F. T.; Burkhart, R. H.; Everson, B. L.; Melvin, R. G.; Young, D. P.; Erickson, L. L.; Madson, M. D.; and Woo, A. C.: TRANAIR: A Computer Code for Transonic Analyses of Arbitrary Configurations. AIAA Paper 87-0034, Jan. 1987.

[38] Everson, B. L.; Bussoletti, J. E.; Johnson, F. T.; Samant, S. S.; Erickson, L. L.; and Madson, M. D.: TRANAIR and its NAS Implementation. Paper presented at the NASA Conference on "Supercomputing in Aerospace," NASA Ames Research Center, March 10-12, 1987.

[39] TRANAIR Computer Code (Theory Document). NASA Contract Report NAS2-11851, Boeing Military Airplane Company, 1987.

[40] EM-TRANAIR: A Computer Program for the Solution of Maxwell's Equations in Three Dimensions: Volume 1, Theory Manual. AFWAL-TR-87-3082, volume 1, 1987. (Limited to U.S. Government agencies and contractors.)

[41] Samant, S. S.; Bussoletti, J. E.; Johnson, F. T.; Melvin, R. G.; and Young, D. P.: Transonic Analysis of Arbitrary Configurations using Locally Refined Grids. Proceedings of the 11th International Conference on Numerical Method in Fluid Dynamics, 1988, pp. 518-522.

[42] Young, D. P.; Melvin, R. G.; Bieterman, M. B.; Johnson, F. T.; Samant, S. S.; and Bussoletti, J. E.: A Locally Refined Rectangular Grid Finite Element Method. Report SCA-TR-108-R1, Boeing Computer Services, Seattle, Washington, 1989.

[43] Young, D. P.; Melvin, R. G.; Johnson, F. T.; Bussoletti, J. E.; Wigton, L. B.; and Samant, S. S.: Application of Sparse Matrix Solvers as Effective Preconditioners. SIAM J. Sci. Stat. Comput., vol. 10, no. 6, 1989, pp. 1186-1199.

[44] Bussoletti, J. E.; Johnson, F. T.; Young, D. P.; Melvin, R. G.; Burkhart, R. H.; Bieterman, M. B.; Samant, S. S.; and SenGupta, G.: TRANAIR Technology: Solutions for Large PDE Problems. Solution of Superlarge Problems in Computational Mechanics, J. H. Kane and A. D. Carlson, eds., Plenum Press, New York, 1989, pp. 95-124.

[45] Johnson, F. T.; Samant, S. S.; Bieterman, M. B.; Melvin, R. G.; Young, D. P.; Bussoletti, J. E.; and Madson, M. D.: Application of the TRANAIR Rectangular Grid Approach to the Aerodynamic Analysis of Complex Configurations. AGARD-CP-464, 1989, pp. 21.1-21.12.

[46] Melvin, R. G.; Bieterman, M. B.; Young, D. P.; Johnson, F. T.; Samant, S. S.; and Bussoletti, J. E.: Local Grid Refinement for Transonic Flow Problems. Proceedings of the Sixth International Conference on Numerical Methods in Laminar and Turbulent Flow, Volume 6, Part 1, C. Taylor, P. Gresho, R. L. Sani, and J. Häuser, eds., Pineridge Press, 1989, pp. 939-950.

[47] Bieterman, M. B.; Bussoletti, J. E.; Hilmes, C. L.; Johnson, F. T.; Melvin, R. G.; Samant, S. S.; and Young, D. P.: Solution Adaptive Local Rectangular Grid Refinement for Transonic Aerodynamic Flow Problems. Report ECA-TR-126, Boeing Computer Services, Seattle, Washington, 1989. Proc. 1989 GAMM Conference on Numerical Methods in Fluid Mechanics, Delft, The Netherlands, September 1989 in Notes on Numerical Fluid Mechanics, Volume 29, Vieweg Verlag, 1990.

[48] Young, D. P.; Melvin, R. G.; Bieterman, M. B.; Johnson, F. T.; and Samant, S. S.: Global Convergence of Inexact Newton Methods for Transonic Flow. Report ECA-TR-124-R1, Boeing Computer Services, Seattle, Washington, 1989.

[49] Chen, A. W.; Curtin, M. M.; Carlson, R. B.; and Tinoco, E. N.: TRANAIR Applications to Engine/Airframe Integration. AIAA Paper 89-2165, July 1989.

[50] Goodsell, A. M.; Madson, M. D.; and Melton, J. E.: TranAir and Euler Computations of a Generic Fighter Including Comparisons with Experimental Data. AIAA Paper 89-0263, Jan. 1989.

[51] Tseng, W.; Feinberg, E.; and Cenko, A.: TRANAIR Applications to Fighter Configurations. AIAA Paper 89-2220, July 1989.

[52] Bateman, H.: Irrotational Motion of a Compressible Inviscid Fluid. Proc. National Academy of Sciences, Volume 16, 1930, p. 816.

[53] Saad, Y.; and Schultz, M. H.: GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM J. Sci. Stat. Comput., vol. 7, no. 3, 1986, pp. 856-869.

[54] Wigton, L. B.; Yu, N. J.; and Young, D. P.: GMRES Acceleration of Computational Fluid Dynamics Codes. AIAA Paper 85-1494, July 1985.

[55] TRANAIR User's Manual, NASA Contract NAS2-12513, The Boeing Company, Oct. 1989.

[56] Weiser, A.: Local-Mesh, Local-Order, Adaptive Finite Element Methods with A Posteriori Error Estimators for Elliptical Partial Differential Equations. Yale University Department of Computer Science Technical Report 213, 1981.

[57] Samet, H.: The Quadtree and Related Hierarchical Data-Structures. Computing Surveys, vol. 16, no. 2, 1984, pp. 187-260.

[58] Strang, G.; and Fix, G. J.: An Analysis of the Finite Element Method. Prentice Hall, Englewood Cliffs, N.J., 1973.

[59] Dembo, R. S.; Eisenstat, S. C.; and Steihaug, T.: Inexact Newton Methods. SIAM J. Num. Anal., vol. 19, no. 2, 1982, pp. 400-408.

[60] Bank, R. E.; and Rose, D. J.: Global Approximate Newton Methods. Numerische Mathematik, vol. 37, no. 2, 1981, pp. 279-295.

[61] Babuška, I.; and Miller, A.: *A-posteriori* Error Estimates and Adaptive Techniques for the Finite Element Method. Tech. Note BN-968, Institute for Physical Science and Technology, University of Maryland, June 1981.

[62] Babuška, I.; Zienkiewicz, O. C.; Gago, J.; and de A. Oliveira, E. R., eds.: Accuracy Estimates and Adaptive Refinements in Finite Element Computations, John Wiley & Sons, New York, 1986.

[63] Bank, R. E.: Analysis of a Local *A posteriori* Error Estimate for Elliptic Equations. Accuracy Estimates and Adaptive Refinements in Finite Element Computations, John Wiley & Sons, New York, p. 119.

[64] Bank, R. E.: The Efficient Implementation of Local Mesh Refinement Algorithms. Adaptive Computational Methods for Partial Differential Equations, I. Babuška, J. Chandra, and J. E. Flaherty, eds., SIAM Publications, 1983, pp. 74-81.

[65] Berger, M. J.; and Jameson, A.: Automatic Adaptive Grid Refinement for the Euler Equations. AIAA Journal, vol. 23, no. 4, 1985, pp. 561-568.

[66] Dannenhoffer, J. F., III; and Baron, J. R.: Grid Adaptation for the 2-D Euler Equations. AIAA Paper 85-0484, Jan. 1985.

[67] Löhner, R.; Morgan, K.; and Zienkiewicz, O. C.: Adaptive Grid Refinement for the Compressible Euler Equations. Accuracy Estimates and Adaptive Refinements in Finite Element Computations, John Wiley & Sons, New York, p. 281.

[68] Oden, J. T.; Strouboulis, T.; and Devloo, P.: Adaptive Finite Element Methods for High-Speed Compressible Flows. Inter. J. Numer. Meth. in Fluids, vol. 7, no. 11, 1987, pp. 1211-1228.

[69] Zienkiewicz, O. C.; Xi-Kui, L.; and Nakazawa, S.: Iterative Solution of Mixed Problems and the Stress Recovery Procedures. Communications in Applied Numerical Methods, vol. 1, no. 1, 1985, pp. 3-9.

[70] Bramble, J. H.; and Schatz, A. H.: Higher Order Local Accuracy by Averaging in the Finite Element Method. Mathematics of Computation, vol. 31, no. 137, 1977, pp. 94-111.

[71] Clark, C. C.; and Foutch, D. W.: PARC Analysis of an Axisymmetric Turbofan Nozzle. Boeing Commercial Airplanes Technical Report PROP-BN31U-C89-020, 1989.

[72] Cooper, G. K.: The PARC Code: Theory and Usage. Arnold Engineering Development Center Technical Report AEDC-TR-87-24, 1987. (Distribution limited to Department of Defense.)

[73] Shankar, V.; Szema, K.; and Bonner, E.: Full Potential Method for Analysis/Design of Complex Aerospace Configurations. NASA CR-3982, 1986.

[74] Shankar, V.; Szema, K.; and Chakravarthy, S.: Supersonic Flow Computations Over Aerospace Configurations Using an Euler Marching Solver. NASA CR-4085, 1987.

[75] Shapiro, A. H.: The Dynamics and Thermodynamics of Compressible Fluid Flow, Ronald Press Company, 1953.

[76] Liepmann, H. W.; and Roshko, A.: Elements of Gasdynamics, John Wiley & Sons, 1957.

[77] Hafez, M.: Progress in Finite Element Techniques for Transonic Flows. AIAA Paper 83-1919, July 1983.

[78] Babuška, I.; and Dorr, M. R.: Error Estimates for the Combined h and p Versions of the Finite Element Method. Numerische Mathematik, vol. 37, no. 2, 1981, pp. 257-277.

[79] Babuška, I.; and Rheinboldt, W. C.: Adaptive Finite Element Processes in Structural Mechanics. Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, 1984, pp. 345-377.

[80] Patera, A. T.: A Spectral Element Method for Fluid Dynamics: Laminar Flow in a Channel Expansion. J. Computational Physics, vol. 54, no. 3, 1984, pp. 468-488.

[81] Fischer, P. F.; Ronquist, E. R.; and Patera, A. T.: Parallel Supercomputing–Methods, Algorithms and Applications. John Wiley & Sons, 1988.

[82] Saaris, G. R.; Gilkey, R. D.; Smit, K. L.; and Tinoco, E. N.: Transonic Analysis of Complex Configurations Using TRANAIR Program. SAE Paper 892289, 1989.

[83] Fraenkel, L. E.: On Corner Eddies in Plane Inviscid Shear Flow. J. Fluid Mech., vol. 11, no. 3, 1961, pp. 400-406.

[84] Chorin, A. J.: Estimates of Intermittency, Spectra, and Blow-up in Developed Turbulence. Communications on Pure and Applied Mathematics, vol. 34, Nov. 1981, pp. 853-866.

[85] Johnson, F. T.; Bussoletti, J. E.; Woo, A. C.; and Young, D. P.: A Transonic Rectangular Grid Embedded Panel Method. Advances in Computational Transonics, Pineridge Press Ltd., Swansea, Wales, 1984.

[86] George, K. P.; Ravichandran, K. S.; Rangarajan, R.; and Desai, S. S.: Vortex Simulation in Full Potential Solver on a Cartesian Grid. Aeronautical Development Agency and National Aeronautical Laboratory, Bangalore, India, 1988. (To be published.)

[87] McLean, J. D.; and Matoi, T. K.: Shock/Boundary-Layer Interaction Model for Three-Dimensional Transonic Flow Calculations. Turbulent Shear Layer/Shock-Wave Interactions, 1986, pp. 311-321.

[88] Chandra, R.: Conjugate Gradient Methods for Partial Differential Equations. Yale University Research Report 129, 1978.

[89] James, R. A.: The Solution of Poisson's Equation for Isolated Source Distributions. J. Computational Physics, vol. 25, no. 2, 1977, pp. 71-93.

[90] Buneman, O.: Analytic Inversion of the Five-Point Poisson Operator. J. Comput. Phys., vol. 8, no. 3, 1971, pp. 500-505.

[91] Rabiner, L. R.; and Gold, B.: Theory and Application of Digital Signal Processing. Prentice-Hall, 1975.

[92] Hurd, A. E.: and Loeb, P. A., eds.: Introduction to Nonstandard Analysis, Pure and Applied Mathematics Series, Vol. 181. Academic Press, 1985.

[93] Zygmund, A.: Trigonometric Series, Vol. II. Cambridge University Press, 1968.

[94] Swarztrauber, P. N.: The Methods of Cyclic Reduction, Fourier Analysis, and the FACR Algorithm for the Discrete Solution of Poisson's Equation on a Rectangle. SIAM Review, vol. 19, no. 3, 1977, pp. 490-501.

[95] Cooley, J. W.; Lewis, P. A. W.; and Welch, P. D.: The Fast Fourier Transform Algorithm: Programming Considerations in the Calculation of Sine, Cosine, and Laplace Transforms. J. Sound and Vibration, vol. 12, no. 3, 1970, pp. 315-337.

[96] Temperton, C.: Direct Methods for the Solution of the Discrete Poisson Equation: Some Comparisons. J. Comp. Physics, vol. 31, no. 1, 1979, pp. 1-20.

[97] George, A.; and Liu, J. W. H.: Computer Solution of Large Sparse Positive Definite Systems. Prentice Hall, Englewood Cliffs, N.J., 1981.

[98] Knuth, D. E.: The Art of Computer Programming, Volume 3, Sorting and Searching, Addison-Wesley, 1973.

[99] Pissanetzky, S.: Sparse Matrix Technology, Academic Press, 1981.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Contractor Report

4. TITLE AND SUBTITLE
TranAir: A Full-Potential, Solution-Adaptive, Rectangular Grid Code for Predicting Subsonic, Transonic, and Supersonic Flows About Arbitrary Configurations—Theory Document

5. FUNDING NUMBERS
C NAS2-12513
WU 505-61-21

6. AUTHOR(S)
F. T. Johnson, S. S. Samant, M. B. Bieterman, R. G. Melvin, D. P. Young, J. E. Bussoletti, and C. L. Hilmes

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Boeing Military Airplane Company
P. O. Box 3707, M/S 7K-06
Seattle, WA 98124-2207

8. PERFORMING ORGANIZATION REPORT NUMBER
A-90093

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Ames Research Center
Moffett Field, CA 94035-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR-4348

11. SUPPLEMENTARY NOTES
Point of Contact: M. Madson, Ames Research Center, MS 227-2, Moffett Field, CA 94035-1000
(415) 604-3621

12a. DISTRIBUTION/AVAILABILITY STATEMENT
[redacted]
Subject Category – 02

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

A new computer program, called TranAir, for analyzing complex configurations in transonic flow (with subsonic or supersonic freestream) has been developed. This program provides accurate and efficient simulations of nonlinear aerodynamic flows about arbitrary geometries with the ease and flexibility of a typical panel method program.

The numerical method implemented in TranAir is described in this report. The method solves the full potential equation subject to a set of general boundary conditions and can handle regions with differing total pressure and temperature. The boundary value problem is discretized using the finite element method on a locally refined rectangular grid. The grid is automatically constructed by the code and is superimposed on the boundary described by networks of panels; thus no surface fitted grid generation is required. The nonlinear discrete system arising from the finite element method is solved using a preconditioned Krylov subspace method embedded in an inexact Newton method. The solution is obtained on a sequence of successively refined grids which are either constructed adaptively based on estimated solution errors or are predetermined based on user inputs. Many results obtained by using TranAir to analyze aerodynamic configurations are presented. (See User's Manual, NASA CR-4349.)

14. SUBJECT TERMS	15. NUMBER OF PAGES
Transonic aerodynamics, Rectangular grids, Complex geometry	264
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102